5-11-2016 12:00 AM

Coordinated Autonomic Managers for Energy Efficient Date Centers

Forough Norouzi
The University of Western Ontario

Supervisor
Michael A. Bauer
The University of Western Ontario

Graduate Program in Computer Science
A thesis submitted in partial fulfillment of the requirements for the degree in Doctor of Philosophy

Abstract

The complexity of today's data centers has led researchers to investigate ways in which autonomic methods can be used for data center management. Autonomic managers try to monitor and manage resources to ensure that the components they manage are self-configuring, self-optimizing, self-healing and self-protecting (so called "self-*" properties). In this research, we consider autonomic management systems for data centers with a particular focus on making data centers more energy-aware. In particular, we consider a policy based, multi-manager autonomic management systems for energy aware data centers. Our focus is on defining the foundations – the core concepts, entities, relationships and algorithms - for autonomic management systems capable of supporting a range of management configurations. Central to our approach is the notion of a "topology" of autonomic managers that when instantiated can support a range of different configurations of autonomic managers and communication among them. The notion of "policy" is broadened to enable some autonomic managers to have more direct control over the behavior of other managers through changes in policies. The ultimate goal is to create a management framework that would allow the data center administrator to a) define managed objects, their corresponding managers, management system topology, and policies to meet their operation needs and b) rely on the management system to maintain itself automatically. A data center simulator that computes its energy consumption (computing and cooling) at any given time is implemented to evaluate the impact of different management scenarios. The management system is evaluated with different management scenarios in our simulated data center.

Keywords

Autonomic management, policy based management, data center

Acknowledgments

I would like to express my special appreciation and thanks to my advisor Professor Dr. Michael A. Bauer, you have been a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow as a research scientist. Your advice on both research as well as on my career have been priceless.

A special thank to my best friend and my beloved husband, Taha, who always is there for me. I would like to thank my family: my father, my mother, and my brother and sisters for supporting me spiritually throughout writing this thesis and my life in general. Last but not the least I would like to thank my young man Kian, during these years I was a part time mom.

Table of Contents

List of Tables

List of Figures

List of Appendices

Chapter 1

Introduction

Data centers at the core of Internet-scale applications consume about 1.3% of the worldwide electricity supply and this fraction is predicted to be 8% by 2020 [1]. Google alone, for example, consumed $2.26*10^6$ MW in 2010 [3]. Carbon emissions from data centers alone in November 2008 were 0.6% of the global total and predicted to be 2.6% by 2020, which is more than the total carbon emission of Germany [2]. Given these statistics, reducing energy consumption of data centers and making them work in *energy-aware* manners is a central topic in research into data center management.

There is broad range of research efforts investigating techniques and approaches to try to minimize energy consumption. Broadly, research in the *energy optimized data center* field could be categorized as follows:

1) Server level energy management: Taking advantage of several power/performance states defined in components, e.g. CPU and memory.
2) Cluster level management: Using optimization and control approaches to optimize the number of required compute nodes for running an application.
3) Virtualization: Reducing the number of active physical servers by multiplexing them as virtual machines (VM) with the aim of using fewer physical servers and taking advantage of turning off underutilized servers. Another aspect of virtualization is considering VM migration and consolidation based on thermal output.
4) Scheduling: Job scheduling that can take into consideration energy consumption criteria, for instance, the temperature of servers, electricity prices and CO2 emission in the case of geographically distributed data center.
5) Using renewable energy sources.

Research in each of these categories is trying to address a part of the energy management of a *complex distributed syste*m, i.e., a data center. To date, there is little research

involving holistic approaches that look at trying to optimize data center energy consumption based on overall governing strategies. The complexities of data centers do not make it feasible for individuals to manage all aspects; instead data centers need automated methods to be able to make management decisions about their operations. In this regard, one broad approach to consider is autonomic management approach to help automate the management of a data center, particularly, policy-based autonomic management, where the role of the administrator would be codifying management policy for data center operations.

Autonomic Computing (AC) [13] addresses self-management in distributed and complex systems. AC tries to minimize administrator intervention in system management by, instead, automating as much of the management activities as possible and enabling the administrator to define the overall policy and strategy for system management according to business and operational objectives. Self-management based on defined policies is called *policy-based management*; it is a promising approach for developing autonomic management in complex distributed systems.

In this research, we advocate for *multiple autonomic managers* rather than having a single centralized autonomic manager that can be a single point of failure and a potential performance bottleneck in a distributed environment. Our research focuses on how to develop a management model based on a number of defined autonomic managers which can be connected together with any defined topology. The management system aims to reduce energy consumption of the data center while still maintaining adequate throughput levels.

1.1 Motivation

Managing today's data centers has become increasingly complex with large scale applications, heterogeneous computing platforms, varying workloads, and service level agreements. This complexity has led researchers to investigate approaches where many of the aspects of data center management can be automated and the day-to-day tasks of system administrators simplified. While data center administrators continue to face

increasing challenges, the operators of data centers must deal with energy and costs and, hence, energy management. While there has been and continues to be research into means of reducing energy consumption in data centers, the overall challenge is to balance the efficiency of computing with energy conservation.

1.2 Problem statement

In this research, we consider the use of policy-based autonomic management for data center management systems. Our goal is to develop a framework for management systems – an abstraction - that can be used to define and create (instantiate) management systems comprised of diverse autonomic managers. Using the model, or more precisely, the associated tools, specifications, etc., the data center administrator would be able to instantiate a specific management system, including the managers to deploy and their configuration according to the physical layout of the data center, the applications, etc.

The challenge is diversity: different data centers will have different computing equipment in different configurations, there are different applications, workloads and, for management, different autonomic managers and ways in which the managers could interact. For example, for a given data center the administrator might define a hierarchical arrangement of managers or perhaps a peer-to-peer arrangement or even an ad hoc combination of these two. Obviously, in some circumstances one of these arrangements may work better than others. We aim to define an abstraction of a management system that gives the administrator the ability to define different management topologies for a given data center. With this abstraction as a foundation, we can define basic algorithms for the deployment and operation of an instantiated management system.

Our approach to management is, as noted, based on autonomic managers. Managers, in turn, need information about the entities they must manage, i.e., the managed objects. Managed objects are defined based on the actual entities in the data center. We formalize these concepts through a model of a data center.

Central to the framework, however, are the notions of "classes of managed objects" and "classes of autonomic managers". This allows for abstraction away from the specific details of objects. Based on the classes, we define a "topology" of management classes which imposes constraints on which managers are to communicate with which other managers and which can be used to introduce a variety of different management configurations.

Figure 1-1 illustrates the abstract entities of our envisioned framework for management systems. Configuration files (defining the data center, managed objects and their classes, managers and their classes, topological constraints) are inputs to the management system.

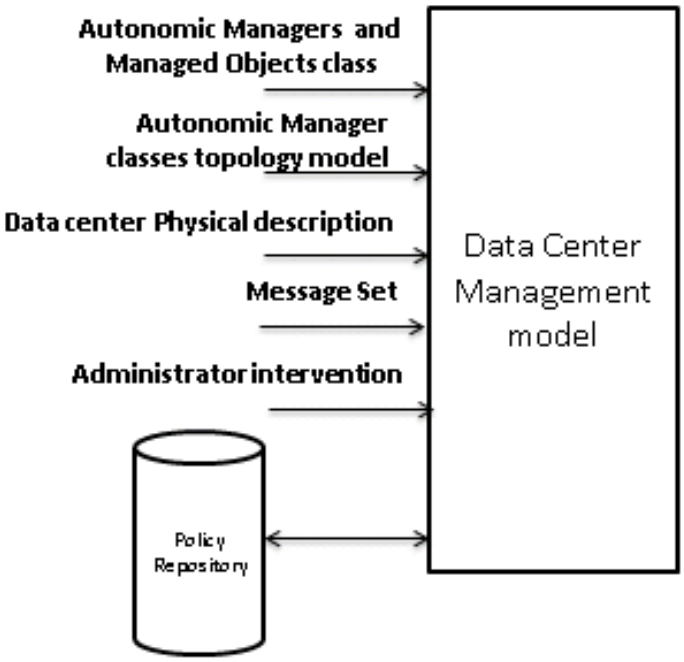

Figure 1-1. Abstract Model of envisioned management system

The definition of the details of objects, etc. in the framework allows us to address other necessary aspects of the actual management system: the way they communicate, the time for communication, and, finally, the associated algorithms for initialization, ongoing operation; we define these as well.

1.3 Contributions

The main contributions of this research are the development of an abstract framework, and associated concepts and algorithms, that can be used to specify an autonomic management system for diverse data centers that is able to support, deploy and manage different configurations of autonomic managers.

In the proposed management system model, a data center administrator is able to define managed object classes with their internal configuration parameters and for each class of managed objects, defining whether there needs to be a manager or not and, if a manager is needed, then defining the manager for the managed object class. Connections between managers are defined via the specification of a management topology defined among classes of managers. In our model we also consider relations among managers where some managers may have privileges over other managers in the system. Privileged managers may have more or broader information in comparison to their subordinate managers and may be able to impose changes on the other managers.

Basic algorithms to set up the management system and initialize the managers have been defined as part of this research. Moreover, algorithms to handle messages, handle events, initiate and terminate the operation of the managers are also presented in this research.

Secondary contribution of this research is the development of a data center simulator that is able to run web based and HPC jobs. To the best of our knowledge, this simulator is unique since it computes cooling and energy consumption of a data center and handles the simulation of running different types of applications, e.g. HPC, web- based. This simulator is available in Github [45].

The main goal of this thesis is the development of a management system built on policy-based autonomic computing concepts for data centers. The proposed management system should work with minimum administrator intervention. To evaluate proposed management system, we need to have reliable and configurable environment that is able to simulate the behavior of a data center. This leads us to the secondary output of this

thesis, which is developing a data center simulator. The simulator can be configured with different data center layouts and workloads.

1.4 Road map

The focus of this research is on developing a generic framework for autonomic management systems for data centers trying to minimize data center energy consumption while maximizing the throughput. The administrator defines the managed objects, classes, their corresponding managers and their classes and a topology of the managers, afterwards, the management system initializes itself and takes care of it on-the-fly. To develop the management model, several issues need to be addressed:

1. How do we model a data center?
   -What are the managed objects from static and physical object to workload in a data center?
   -What type of applications run in a data center and how to describe them?
2. How to model the managers?
   -What is the granularity of managers? (i.e. which object/s need to have manager).
   -What are possible actions that a manager might be able to do to adjust the behavior of its managed object/s?
   -What kind of information should a manager know about the managed object/s?
3. How are the configurations of autonomic managers specified?
   -How to specify the topology of the managers?
   -How is communication between autonomic managers done?
4. How to deploy the management system model (initialization, management cycle)?

Chapter 2 provides a review of related work in the areas of data center energy consumption, autonomic computing, and policy based management. Chapter 3 covers general aspects of a data center and its management. Chapter 4 describes the formal model of data center, and managed objects and their classes. Chapter 5 defines abstract model of the management system i.e. managers, their classes, connections between

managers and topology of the management system. Algorithms for deployment and operations of the of a management system in Chapter 6; these algorithms must ensure that constraints created using the framework for specifying a management system are enforced. In Chapter 7 the performance of the management system is illustrated using different scenarios which are executed and evaluated using our own data center simulator. Conclusion and future aspects are considered in Chapter 8.

Chapter 2

Related Work

With the objectives of our research in mind, related work can be categorized into different categories:

- Previous research on approaches for data center energy conservation;
- Autonomic computing and policy based management;
- Coordination and collaboration of computational agents;

Review related works in these areas are discussed in the following.

2.1. Data center energy conservation

These days data centers use raised floors and lowered ceilings for cooling air circulation, with the computing equipment organized in rows of racks arranged in an aisle-based layout, typically done with interleaved cold aisles and hot aisles (see Figure 2-1). Server racks may have chiller doors that function as radiators to cool down hot air coming out of servers. The cooling of the data center room is commonly done through computer room air conditioning (CRAC) units. In this case, cool air comes into the data center through raised floor vents [18]. More recent designs have racks of computers cooled by liquids that are pumped through the racks, servers and even chips. There may or may not be CRAC units as well.

Data centers consume electrical power mainly for their server operations (computing power) and for the cooling system, which removes heat generated by running servers. Power Usage Efficiency (PUE) is a data center management parameter that indicates the energy efficiency of a data center. PUE is defined as total power consumed divided by power consumed by the computational equipment; where total power [30] is the sum of the power used for computational equipment plus the power used for cooling. Generally, normal data centers have PUE value around 2, however newer data centers are much

more efficient; to date the greenest data center is owned by Facebook [29] with a PUE of 1.07.

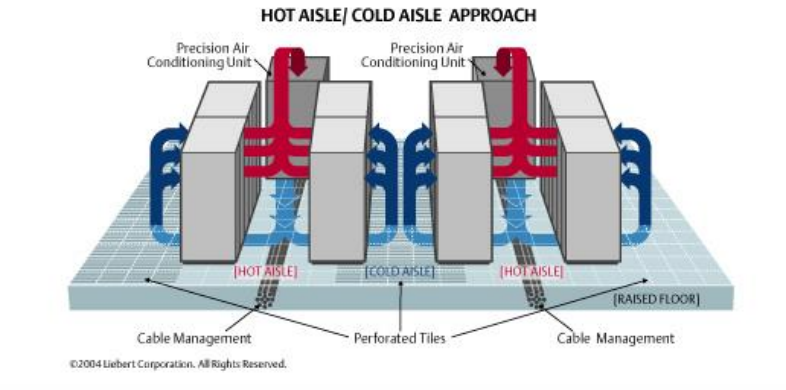

Figure 2-1. Data center thermal aspect architecture [15].

Approaches to manage or control energy consumption in data centers have encompassed broad areas of research - from managing dynamic changes in hardware settings in servers to developing management frameworks for whole clusters of servers and complete data centers. Previous studies can be classified into several categories: 1) options for controlling power: dynamic provisioning of servers by turning off idle servers and turning on servers on demand, scaling the voltage or frequency of servers, creating idle servers by using virtual machines and consolidating them; 2) optimization approaches that try to use compromise between user satisfaction and power budget; 3) methodologies that deal with scheduling and workload, such as thermal workload scheduling, virtual machine migration, workload prediction, 4) scheduling to take advantage of renewable energy, like wind to run data center. We review research in these areas in the following.

Many early solutions for reducing energy consumption focused on managing a single node server by using Dynamic Voltage and Frequency Scaling (DVFS) [27]. The main assumption for this idea is that CPU is the main sink for absorbing power in a server [26] Other research [27] has shown that there is a linear relationship between power consumption of a CPU and its utilization (refer to Figure 2-2).

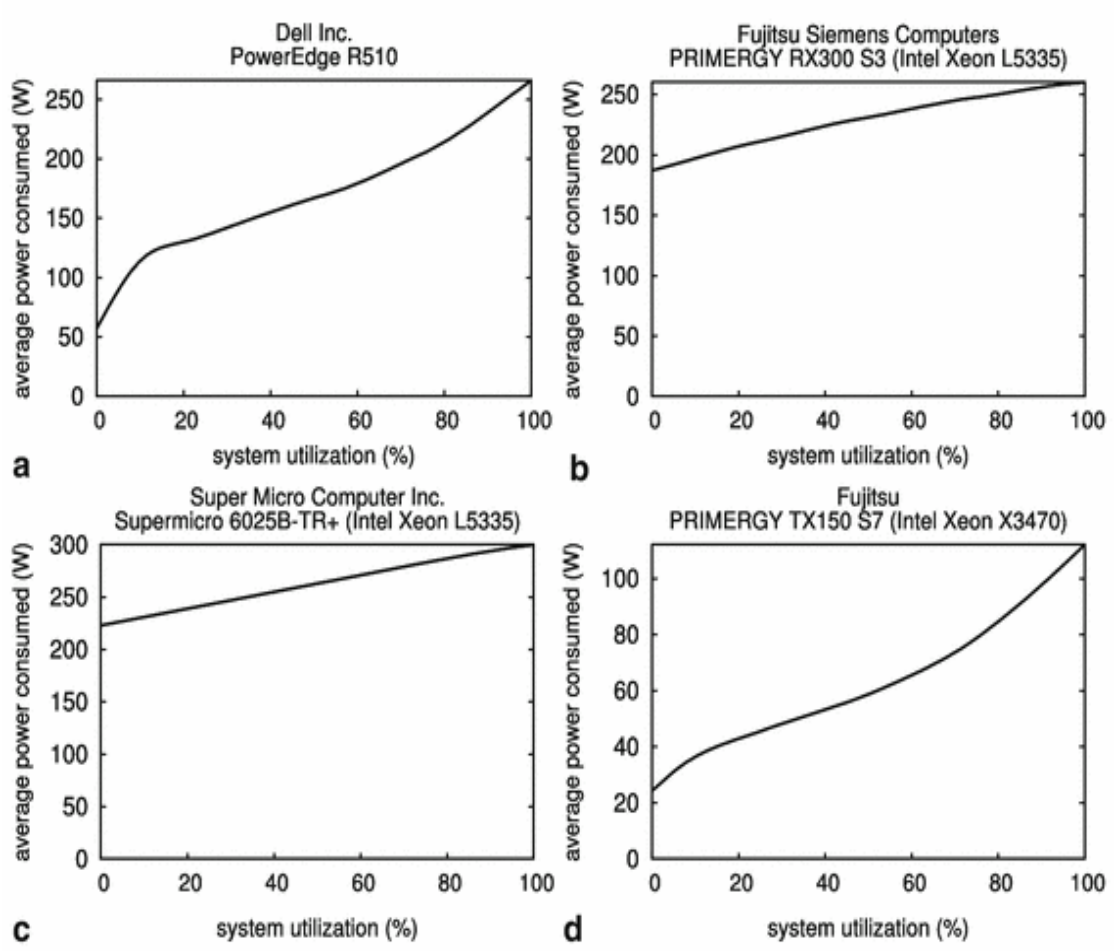

Figure 2-2. Power Profile for various server platform [44].

The use of low power consumption levels of the hardware idle state (CPU and memory) is another target of research for reducing energy consumption. In [24], Meisner et.al. have introduced a new state for the whole server i.e. called the ***Power Nap:*** minimum power usage for all components in the server like laptops that support ACPI[1] in which the CPU is in sleeping state and self-refresh state for DRAM. By having power nap, rather than having components with fine-grain power and performance trade off, the focus would be designing just one low power state and one high performance active state where the main consideration would be fast transition between these two states. In their paper, they illustrate requirements and mechanisms to develop the Power Nap concept in high

[1] *Advanced Configuration and Power Interface*

density blade server systems. They collected different traces of several servers with different workloads. They observed that the majority of idle periods are shorter than 1 second with the mean length hundreds of milliseconds (ms) and busy periods are even shorter than 100ms for some workloads. They tried to estimate the characteristics of the hardware suitable for the idea i.e. leveraging the short idle period due to workload change. They constructed a queuing model based on characteristics of their traces. They found that if the transition time was less than 10ms, then power saving varies linearly with utilization. But with the increase in the transition time, the power saving and performance decrease. The problem is that with current hardware technology, achieving transition time in less than 10ms is hard to achieve.

Data center provisioning algorithms attempt to provide the number of servers that guarantee the response time specified in service level agreements, which often specifies the maximum load. In practice, most of the times data centers are operated far less than their maximum load, e.g. 30-70% of their maximum load [31]. The over provisioning causes a lot of servers with no load or partial load to run, which consume power. Some research has looked at server elasticity based on the volume of predicted workload and trying to consolidate all the workload in a number of servers at a given time [32][50] and powering off unused servers or at least making them work in low power mode.

As another effort in this approach, Chen et al. [50] and Chen et.al. [17] proposed dynamic server provisioning for long-lived TCP-based services. They used data traces of Windows Live Messenger, and built a prediction model for estimating the number of required servers, (every 30 minutes). Their algorithm saves energy by turning off the unnecessary servers. They balance the load among active servers. While this approach reduces power consumption it has its own concerns: increasing cooling power since by having number of servers with maximum utilization causes hot spots (even if the server are chosen to be distributed across the data center) and this is usually not desirable for cooling systems. Another concern is degradation in response time, since peak workloads need to wait until new serves become active. A combination of active server provisioning and workload prediction could provide better results.

Another approach to decrease number of active servers is through the use of virtualization. By moving services to virtual machines, several servers can be time-multiplexed on a physical machine and increase the CPU utilization of that physical machine. Basically, through virtualization, the number of running physical machine can be reduced and this reduces energy consumption [33][24][56]. Kephart et.al [38] have investigated a power and performance efficient resource management in a virtualized computing environment. Specifically, they try to tackle dynamic provisioning of VMs for web applications based on the current state of the workload, which is defined as the number of incoming requests. The SLA for each application is specified as the desired request-processing rate. Clients pay for the provided service and in case of a SLA violation they would get refunded as penalty by the data center providers. The main idea is to maximize the provider's profit by minimizing both power consumption and SLA violations. They posed the problem as a sequential optimization and used a limited lookahead control (LLC) to solve it. The goal was to optimize the following parameters: the number of allocated VMs to each service; the CPU share of each VM; the number of active servers. A Kalman filter was used to estimate the future workload. The complexity of solving the optimization problem is a challenge in this approach. With just 15 physical hosts, their algorithm takes 30 munities to run. In a real data center with thousands of physical nodes, resource management needs to be timely.

Some research has looked at workload scheduling strategies to reduce server uses. Thermal aware scheduling looks to determine placement of jobs on servers which are "cool" or has minimum thermal effect on others.[20][31] [32].

Mukherjee et.al and Tang et al. [15][50] have modeled the heat that is circulated among the servers; using this model, they suggest spatio-temporal thermal-aware job scheduling algorithms for HPC batch job data centers. One of their proposed spatial scheduling is the least recirculated heat (LRH). In LRH they rank and sort servers according to how much of their produced heat is recirculated and has thermal affects on others, and they assign the jobs to the low-ranking servers. To develop the thermal aware scheduling algorithms

we need to have a model of the thermal behavior of data center. Obviously, in order to meet the SLA requirements, the number of active servers must be carefully chosen. The main challenge is the determination of each application's resource demands.

Chase et.al. [33] have applied an economic framework in the way that management system allocates resources to a service in order to maximize the profit from the service based on the cost of resource and amount of estimated utility. Each service bids for resources. The cost for the resource here is energy cost.

Another type of research on energy aware data centers can be called ***environmentally-conscious*** which assumes that there are number of data centers which are geographically distributed over the globe. The idea is that in job scheduling, the system considers the price of electricity, CO2 emission rate, and amount of renewable energy that can be used in a location and tries to maximize the profit of different locations [34][35].

Google, Microsoft and Yahoo! have started to use forms of the renewable energy resources to power their data centers [37][38].

2.2. Autonomic computing

Autonomic Computing (AC) refers to the idea of a computing system or application being *self managing*, that is, a system that can manage itself in such a way that it is adaptable to any changes in the system environment [21]. In autonomic computing, a management module which controls the behavior of a *managed element* (MO) is called an *autonomic manager* (AM). IBM initially introduced the idea of autonomic management, and suggested that an AM continually goes through a cycle of *monitoring, analysis, planning, and execution* [13]. The idea in AC is that different AMs control different resources in distributed manner. This management could be done individually, i.e., each AM is responsible for its own MOs. More generally, in computing systems it is necessary that AMs interoperate. There may be heterogeneous types of AMs and AMs with different objectives. The research presented in [19] Kephart et.al. demonstrates the coordination between two independent AMs. In this work, the first AM deals with SLA

management and resource allocation to reduce SLA violations. The second AM deals with minimizing power consumption by turning off unused servers. This work showed that without interaction between the managers there might be a failing to achieve their goals.

Another work on the interaction between AMs is presented in [5]. Anthony et.al develop distributed management framework to meet SLA requirements; different AMs interact with one another indirectly and upon the failure of one of the AMs, the others notice the failure via a shared repository and will take over the responsibility of the failed AM. Khargharia et al. [4] introduced a three-level hierarchy for optimizing energy consumption and SLA violations. The hierarchy starts from the device level inside a server, proceeds to the server level and then encompasses the cluster level. Decisions are based on the power status of each managed element at each level. Their idea illustrates the value of a hierarchical approach, but needs some modification to be applicable for large scale data centers which have many different types of applications and services running at the same time and are highly dynamic.

Mola et al. [52] developed a management model for multiple AM collaboration and communication based on their active policies. They considered a hierarchical AM model and developed a message passing communication model between AMs, and also developed algorithms for handling the dynamic nature of AMs: their arrival/initiation, registration, and departure. Evaluation of algorithms is done on Eucalyptus [59] virtual cloud.

Keller et al. [60] introduced a hierarchical approach for dynamic resource management for cloud systems in data centers. They leverage the topology of network to design the management system which makes use of autonomic managers to manage the placement and migration of virtual machines to optimize resources.

In an *autonomic data center,* there may be hundreds of executing AMs with different goals (QoS, energy, security, and configuration). Kennedy et.al. [8] argues that the mechanism that defines interoperability between autonomic elements must be reusable and generic enough to prevent complexities. A standard means must be defined to

exchange context between autonomic elements. They have proposed a social metacognition level for AMs which monitors and controls the AM objectives. For example, this level determines what data should be collected about the MO and how often. This meta level evaluates policies currently being applied by the AM.

Anthony et.al. [5] identify collaboration as a key aspect of and suggest that AMs should be deigned-for-collaboration and that the lack of collaboration is a problem. Then they attempt to tackle the AM interoperability issues and define an interoperability service. The interoperability service keeps a database of registered AMs along with corresponding resources they manage and the scope of their management operation. A standard description language is defined for registered AMs to provide details on their management capabilities. In addition, the interoperability service will detect potential conflicts and send messages to related AMs to, e.g. suspend or stop their activities. The proposed service has a hierarchical architecture, enabling conflict detection at global level (such as system wide security management) and local level (VM level resource management).

2.3 Policy Based Management

Policy Based Management (PBM) is a management paradigm that separates governing rules from the main functionality of the managed system. The aim of PBM is to reduce management cost while making it adaptive. Administrative objectives are defined as policies, so that each AM has its own set of policies specifying how it is to manage its components. According to [12], a policy is a rule that defines a choice in the behavior of a system. A comprehensive review on policy based management system, its standards, and relevant techniques can be found in [12].

A simple policy format is based on the Event-Condition-Action paradigm.

On Event: E1
{
If (Condition: C1…Ci)
Do Action: A1,A2,…,Aj
} with priority Pi

Such a policy means that when the event *E1* occurs, if the set of conditions *(C1,…,Ci)* are verified, and if all policy with priority more than *Pi* have been run then execute the action set *A1,…,Aj* . The policies are provided to the AM at the time that the AM is initialized, but can be changed during the time that an AM is running. This provides a level of dynamic management.

In [22], some of the advantages of policy-based management over hard-coded management have been reviewed. First, PBM eases administration and maintenance (e.g. change). Management policies can be maintained in a repository, reviewed, and changed rather than trying to trace the hard-coded management logic spread across system. Second, the separation of policies and application logic makes for easier implementation since the policy author can focus on modeling policies without considering the specific application implementation. On the other side, the application developers do not have to be worried about how to implement management logic and application developers just need to provide hooks to make their system manageable, i.e. to enable self-management. Third, it is easier to share and reuse the same policy across multiple different applications and to change the policy.

Bahati et al. [54] described architecture for autonomic management and demonstrated how policies are defined and mapped to their corresponding elements. In the context of policy based autonomic management, they categorized policies as: configuration policies, e.g. setting static configuration parameters, expectation policies, i.e. making sure that operational requirements are met, and management policies, i.e. dealing with action and information for managing the management system. They also defined architecture for the policy-based autonomous management system, in particular an Apache server.

Mukhrejee et.al in [18] propose a ***Model-driven Coordinated Management*** architecture to make dynamic management decisions based on the energy benefits of different policies to handle events. They used a workload model, power model, and thermal model to predict the impact of different management policies. They defined a state-based model to dynamically decide the correct management policy to handle events, such as a new

workload arrival or failure of a cooling unit, which can trigger an increase in the ambient temperature. They considered the data center as a state machine, which has two states, normal (no SLA violation) and critical (happening of any critical events, like the ambient temperature reaching the redline temperature). A *central management* unit monitors events, chooses the best policy and makes decisions.

2.4 Collaborative management

One of the main questions arising in the proposed research is developing a model for collaboration and coordination between different autonomic managers; in this regard we dedicate part of literature review on research on collaboration between agents.

Schneeweiss et.al in [41] has categorized coordination between distributed agents (in general and not limited to AMs) that are engaged in a distributed decision making process in different levels:

- ***Data integration***: A smooth exchange of state information between agents, in order to guarantee the data consistency between agents. Each agent makes decisions by considering the updated information they have about each other.
- ***Reactive negotiation***: A market style interaction between agents, which needs to have a coordinator agent, which has superior rights to set and control negotiation rules and facilitate the communication process between all other agents. Every other agent reacts to the behavior of its neighbors based on rules regulated by the coordinator agent.
- ***Integration through planning activities***: Hierarchical-like coordination between agents through top-down and feed forward bottom—up influence. A top-level agent determines top-down effects to agents beneath it. The overall objective of an agent is defined by the top-level agent. This coordination model could be implemented in our proposed research in the way that the top-level AM could affect the behavior of the AMs beneath it.

As seen in Schneeweiss's categories, the nature of coordination between agents increases through the levels. In the first level, agents just share status information and in the last level, one agent can even change the behavior of other around agents.

In [46], Baldassari et.al. described an autonomic cluster management framework. They defined three different types of agents: *general agents* (implemented per node), *optimization agents*, and *configuration agents* (implemented per implementation of the management framework). Each agent has a specific purpose and was designed to collaborate with others to achieve maximum performance of the cluster through load balancing. The proposed management infrastructure is a hybrid of centralized and decentralized management and communication between agents is done via message passing. Generally, they investigated what the four key properties of self-management (self-healing, self-protection, self-optimization, and self-configuration) would be involved in a cluster management system. A drawback of this approach is the complexity of task distribution between agents, which leads to much work for the optimization and configuration agents, which makes them un-scalable for large-scale environments.

Thomas et.al [42] presented a management framework for the automated maintenance cycle in the computing cluster. This project was part of the *Data Grid* project [43]. The management framework tries to address the self-configuration and self-healing features of self-management. A number of management module e.g. job management, monitoring, fault recovery, and configuration management, have been defined where each has information as an output which is used as input for others. Their system gets configuration states from an administrator. On each machine different sensors collect monitoring information periodically; monitoring agents controls sensors. Each machine has a *goal state*, which is stored in a configuration database and also has an *actual state* which comes from the monitoring agent. These states are compared within the *fault detection and recovery system* for any mismatch, which then applies any necessary actions to fix them. The fault detection system has its set of rules (policy) on each node that these rules are checked. The fault detection system is very limited in policy-based functionality. In this work they try to address self-configuration and self-healing properties of general self-management features. Generally, there are a finite number of

hardware and software configuration states. The current state of hardware and software should been checked with these states. At the conceptually high level, there are numbers of non-autonomous management modules pipelined together that the actions are taken by just the installation module and the others just provide proper information. Therefore, there is no coordination between managers.

A decentralized architecture, *Unity*, was introduced in [52]. *Unity* introduces a two level management model that tries to allocate optimized resources (servers) to different types of application environments running both batch type and interactive workload across the whole data center. A management element in each application (first management level) computes a resource level utility function based on the degree of adherence to an application specified SLA. Resource level utility function data from all applications are sent to the second management level (central coordinator), which makes global decision (global utility function) on how much resource should be allocated to each application. Each application manager sends the U(R) resource utility vector to central resource arbiter to inform it the current status of the application. These values are in common values and comparable. They have deployed their framework for a prototype data center with two type of application with different utility function. The first application is transactional workload and second one handles long running batch workload. The resource arbiter allocates proper number of compute server to these applications according to their claimed current utility. The main goal of resource arbiter is to maximizing summation of all utility functions. Their work has scalability issues, since there is one global controller, which decides about all the other local controllers. Adding a new application (new management agent) introduces challenges, e.g. changes in the coefficients of utility functions. The authors even extended their research to encompass optimization over application performance and power consumption, by including a power cost model in the utility function and extending the central coordinator to calculate the optimal number of servers to be off [53].

2.5 Conclusion from the literature review

Research on data center energy management has become an important area. The large scale and dynamic nature of a data center cannot rely on management solely by administrators. Instead, management needs to be done more automatically or, as we propose, more autonomically and leaving administrators to focus on defining behavioral policy. Developing policy based autonomic management framework with focus on energy consumption is cornerstone objective of this research.

A review of previous research in this area suggests that several issues require additional study:

1. What are possible scalable management frameworks that deal with data center energy management and are also capable of dealing with dynamic nature of running systems, servers and applications?
2. What are the characteristics of a management model, i.e. positioning of managers, topology of managers, and AM granularity across a data center?
3. What constitutes an effective communication protocol model among managers all over a data center considering the dynamic changes in systems, i.e., elements to be managed.

In this research we developed a general, configurable, and physical layout independent management system model. We specify the basic requirement for define the model and also major requirement to develop the management system for a data center. In Chapter 6 we investigate the deployment and operation model of the management system.

Generally speaking, the data center administrator as the developer of the management system needs to focus on managed object definition and autonomic manager granularity (where we need to put an autonomic manager), the topology of managers, and communication between managers. Since this research focuses on policy based management system, the corner stone of the model is consistent set of policies for each managed object in the data center.

Chapter 3

Logical Aspects of a Data Center Management Model

There are many different objects in a data center, memory modules, network cards, servers, racks, virtual machines, applications, clusters of servers, etc. These objects need to be managed. In this Chapter we provide an overview of the management requirements of data centers and describe the essential elements of a management model.

A common approach within the autonomic computing paradigm is to define a management component, as an agent, that is attached logically to each object as its manager. In a policy-based approach, each autonomic manager (AM) would have its own policies and a control loop that monitors parameters of the object and enforces the policies in its policy set. The ultimate goal of an AM is to make sure that its policies are enforced. Within the context of a data center, then, we would have multiple autonomic managers to manage diverse elements. These AMs need to interact with each other; to cooperate together in order to meet the objectives of the data center. One can think of the AMs and their relationships as a kind of *management overlay network* on top of the elements of the data center; this is illustrated in Figure 3-1. The actual position of a manager might be on single physical server or even distributed over number of servers.

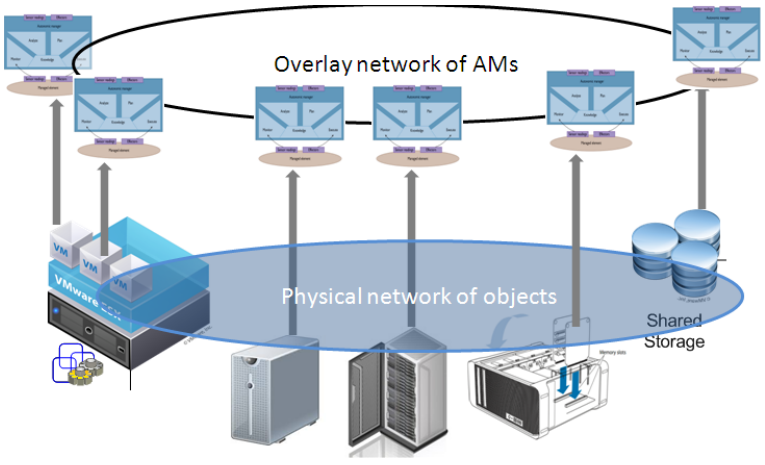

Figure 3-1. Autonomic Manager Overlay.

Key questions that must be addressed include: what metrics about the object should or can be monitored and what are possible actions that the management system could take. In Table 3-1 we illustrate some examples for number of common objects in a data center and identify some possible parameters and actions (keeping in mind that our interest is on energy aware data centers). Ultimately, these parameters and actions are useful for defining policies and steps that autonomic managers can take.

Table 3-1. Example manageable objects and associated monitoring parameters and actions.

Object	Possible monitoring parameters	Possible type of actions
VM	•Throughput •CPU utilization	•Migration •Blocking
RDRAM	• Power state (active, nap, standby, power down)	• Changing power state
Server	• CPU utilization • CPU power state • Server queue job length	• Changing power state • Shutting server down; bringing it up • Invoking admission control
Cluster	• Node utilization • Number of waiting job	• Shutting down nodes • Workload manipulation
Rack	• Number of available nodes • Number of running applications • Number of running system	• Shutting down some nodes • Changing node CPU frequency is applicable
Application	• SLA violations • Number of active allocated servers • Percentage of idle servers • Number of jobs/requests in the application queue	• Make a number of serves idle • Activating a number of servers • Blocking application (stop running and just queueing workload) • Change frequency of allocated servers
Data center	• Current power consumption • Current electricity rate • Current temperature • Data center utilization	• Shutting down racks • Activate racks • Turning on/off a cooler

3.1 Data center management system requirements

A distributed autonomic management framework for a data center entails a number of challenging questions. We review these below and identify those that we focus on in our work:

- *Topology of the Autonomic Managers*: If we view AMs as an overlay network on the physical layout of data center, then in specifying the AMs we need mechanisms that will enable an administrator to specify the topology among the AMs when they are deployed. We will also need algorithms to ensure that the logical connectivity as specified by the administrator. For deployment, there will need to be a specific protocol to enable the AMs to communicate and exchange information e.g. SOAP [14]. There are, of course, a number of questions that arise in the definition of a topology of AMs. How to decompose the management tasks between AMs? Is one AM arrangement better than others? Can part of this be done automatically? These questions are beyond the scope of this thesis, and likely will depend on the specific details and characteristics of each data center, its systems and the applications it runs. Our goal is to provide the means whereby the administrator can specify a particular topology and the management operations to support those activities. However, beyond this model there are possibilities to examine the different management topologies and different management task decompositions.

- *Collaboration Strategies among the Autonomic Managers.* Depending on the topology, the next question is how do AMs influence other AMs in the management system? How much information do they need to share? What kind of information? One AM may be privileged over some set of other AMs because its management scope is wider than the others or it has more information about its surrounding environment. Alternatively, all AMs could be acting the same, e.g. as in a peer-to-peer topology.

- *Life Cycle of Autonomic Managers.* An autonomic manager has its own life cycle, which obviously corresponds to the life cycle of its associated MO/s. For example, for a cloud user renting compute nodes and running an application for a period of time, the corresponding AM is born and dies along with the application life cycle. One of the issues in multilevel management systems is that each level of the management model has

to have the ability to create AMs based on the respective MO life cycle, then introduce it to the management system, and then to destroy it at the end.

Concerns to be addressed are around how an administrator can specify the topology of the autonomic managers, how strategies for coordination/cooperation can be specified and the algorithms needed to handle deployment. We consider this, in particular, in managing a data center where there may be conflicting objectives around ensuring that performance SLAs are met while trying to save energy.

3.2 Topologies and Coordination of Autonomic Managers

If AMs are to have their own overlay network, the topology has implications on the coordination and communication among AMs and the decomposition of management tasks among AMs. Since a data center is dynamic, there will be changes in the applications, systems, etc. that are running and so it is important that such changes are reflected appropriately in the topology of managers to ensure ongoing and consistent management. Generally, one can consider a variety of topologies.

- ***Hierarchical*** means that some AMs can monitor and influence or control the behavior of other AMs. In this case, lower AMs are considered as managed elements for the higher AMs. AMs at different levels usually work at different time scales. In this topology the upper layer AM regulates and orchestrates the system by monitoring parameters of all of its lower level AMs (see Figure 3-2). The upper layer AM is privileged over lower layer AMs, and has the authority to control or manipulate some parameters of the lower level AMs.
- ***Peer to peer*** entails AMs that can directly communicate with one another, exchange information, and have policies that take into account the information of other AMs. In this paradigm, all AMs are often equally privileged.
- ***Indirect coordination*** between AMs involves an AM making changes in its MO/s which these changes are then sensed by other AMs causing them to perform actions. This case needs multiple AMs to be assigned to the MO/s. There is no direct communication between the AMs. Since the MOs (e.g. application, services, and virtual machine) may change over time (e.g. is

finished or started), therefore topologies of corresponding AMs change on the go.

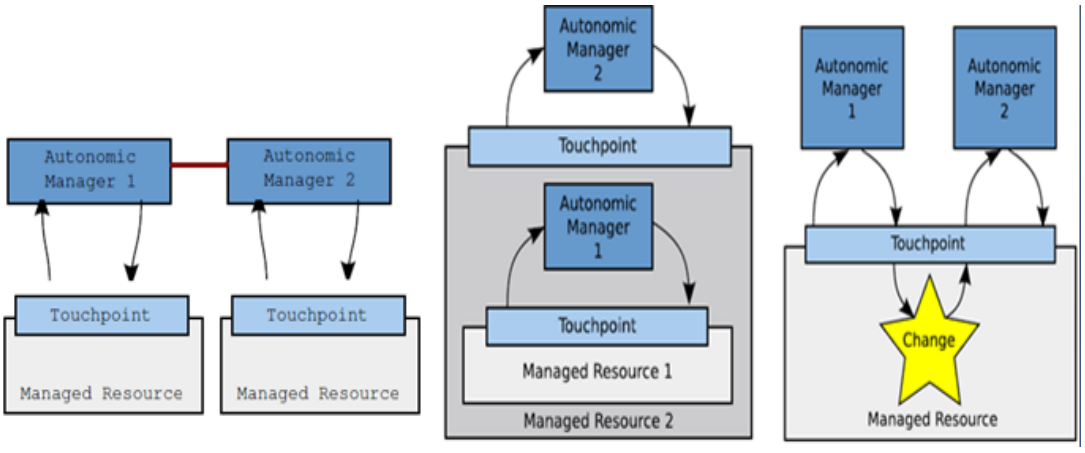

Figure 3-2. AM Coordination: a) Peer to peer, b) Hierarchical, c) Indirect [40].

In our work, we are concerned about topologies of AMs that can support hierarchical and peer-to-peer organizations and variants. We are not interested in supporting indirect types of interactions since these are very dependent upon the specific of the managed objects.

3.3 Management system operating features

An important property of our management system is that it arranges itself in automatic fashion based on information and constraints specified by the administrator. The administrator specifies a number of features, e.g. description for each managed element class, type of cooperation between managers, and configuration of the management system. After the initial start-up phase, the system should be able to configure itself dynamically. To do so, upon receiving a new request for running an application, the central scheduling algorithm needs to initialize a new autonomic manager, create associated operational information, deploy the application and configure the manager appropriately in the management system.

We aim to develop policy based management system, so the system needs to be designed in a way that the administrator can alter policies anytime, and be able to update the policies of any manager at any time within the data center. Communication between the management system and administrator's policies is another required feature.

Considering the above operational and configuration requirements, autonomic managers will need to have a core set of modules: *policy handling*, *event handling*, *decision-making* and *communication*. Managers will need to be able to handle policies, including updating a policy list as well as operating with policies, which will require some form of event handling. Decision-making will need to carry out the analysis of events and policies and decide on actions. Finally, managers will need to collaborate/coordinate among one another and will need to communicate with administrator components, and so a communication component is required.

Chapter 4

Abstract Data Center Model

In this Chapter, we introduce an abstract model of a data center considering different types of systems and workloads. Using this model, we then introduce the model of manageable objects that could fall within the scope of our management model.

4.1. Data center model

We can think of a data center abstractly as consisting of a number of compute nodes, and coolers laid out in some spatial configuration pattern with some network connections among them (multiple separate clusters are each collections of racks, with perhaps no communication between the racks in different clusters). Each rack is comprised of a number of chassis, and within each chassis there are numbers of servers (compute nodes).[2]

On top of this physical infrastructure, we have defined a ***System***; our notion for a number of computes nodes inside a number of racks, which are capable of running the same type of jobs. We assume that systems are defined in terms of a set of racks. All compute nodes inside racks can be shared between different applications that run on that system. At any time, a node inside the system is either assigned to an application, is ready to be assigned to one or is idle (at rest or powered down). We assume that we could have computer nodes that may have different computing power, and that some CPUs may have multiple frequency levels. Managers can use dynamic frequency scaling to manage energy consumption.

---

[2] This is the level of granularity of hardware that we will assume, but it is straightforward to extend the mode to include additional elements, such as network switches, network interface cards, processors, memory, etc.

Application behavior and workloads are key elements in data center operations and have a direct impact on the energy consumed by a system and, hence, the data center. In our model, we consider three broad classes of applications:

1) Interactive Applications: provide access to users across the Internet/intranet, such as web servers, transactional servers, etc. These applications typically process short requests (transactions) and fast response time is main objective of these types of applications. We model this as an *InteractiveSystem*.

2) Enterprise Applications: applications associated with different business units where the applications may require large amounts of secure, reliable data transfer and high availability, running 24/7, e.g. a human resources system. Workloads in these kinds of application vary – from short requests/jobs to much longer activities, e.g. report generation. The key characteristic is that these systems typically run for long periods.

3) High performance computing (HPC) Applications: scientific or other applications which mostly need multiple CPUs to do highly computational tasks.

With these concepts in mind, we can think of a data center as a set of systems running different type of workloads, so our logical view of a data center is modeled as in Figure 4-1.

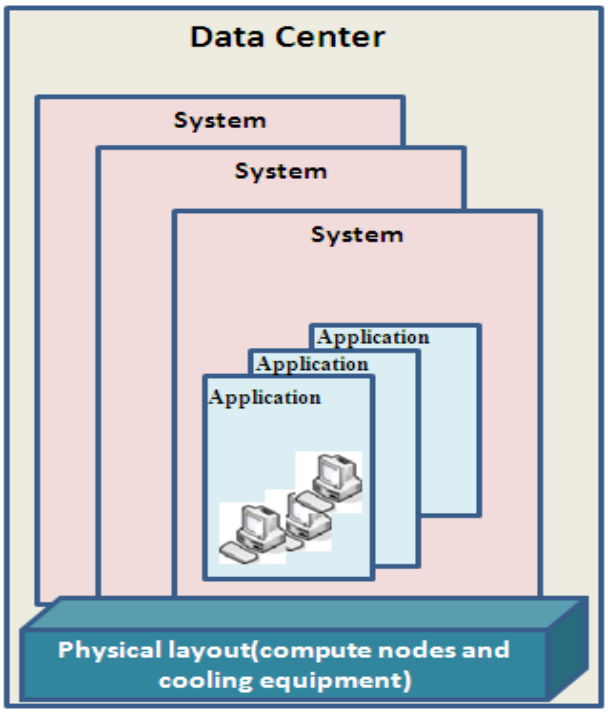

Figure 4-1. Data Center Abstract Model

We make these notions more precise in the following.

Definition 1. *A Data Center, DC, is defined as DC = <Systems, Racks, ThermalModel>*

where:

- *Systems = { Sys_1, Sys , … , Sys_q} is a finite set of systems in the data center;*
- *Racks = {R_1 , R_2 , …, R_k}, where R_i = {r_{i1}, r_{i2} , r_{i2} ,…, r_{ir}};* each R_i is a distinct type of rack, and each rack is a set of chassis r_i= {ch_1, ch_2, ch_3, … ,ch_n} and each chassis ch_j={n_1, n_2, n_3..., n_B} is a set of compute nodes.

Definition 2. *The ThermalModel for a data center is defined as <coolerList, RedTemp, ThermalMap>*

where:

- *coolerList*= {cl_1, cl_2…,cl_n} each cooler is represented as an CoP_Equ equation. Cop_Equ is an equation modeling the coolers. [15] The output of the equation is

the CoP (Coefficient of Performance) parameter of the coolers which indicates how much cooler removes heat by consuming a one Joule heat. Therefore, the higher the COP value is, the more efficient the cooling system is. The input to the CoP equation is the inlet temperature to the cooler.

- *RedTemp*: is an integer that represents the maximum temperature that the data center can tolerate, which depends on type of hardware and servers in the data center.
- *ThermalMap:* is an n*n matrix of number of chassis and each entry determines heat recirculation of that entry to other chassis (see Data Center Energy Consumption).

The Thermal Model models the thermal behavior of the data center. Our thermal model is based on the work illustrated in [15].

4.2. System Definition

A *system* represents a *number of compute nodes running the same type of workload belonging to specific applications or users*. Our data center model defines different types of systems based on general categorization of workloads. Three different types of systems are defined: **enterprise systems**, **interactive systems**, and **high performance computing systems (HPC)**.

Each system needs to have its own resource allocation algorithm in order to allocate nodes to workload and also its own scheduling algorithm for job scheduling. First, we need to define different types of jobs and workloads and the required parameters for representing them.

4.2.1 Jobs and Workloads

We introduce our definitions of jobs and workloads for each of the systems in the following.

HPC jobs require a number of nodes to run and are commonly CPU intensive. The service level agreement (SLA) parameter for each HPC job, if it has one, is its deadline, which is its maximum waiting time before it starts to run plus its required running time.

We define an HPC job as follows.

Definition 3. *An HPC job, h, is defined as <duration, utilization, nodes, deadline>.*

where,

- *duration* is the expected length of execution,
- *nodes* is the number of compute nodes needed,
- *utilization* is average node utilization,
- *terminationTime* is the target time that the job should be in the system more than this time means this job has SLA violation.

In an HPC system we consider an SLA to be violated if a job finishes past its deadline.

We can then define the workload associated with an HPC system as follows:

Definition 4. *An HPC Workload, HPC_W, is defined as $HPC_W = \{(job_1, t_1) (job_2, t_2)_{,\ldots,}\}$*

where

- HPC job_i is an HPC job
- t_i is its arrival time.

Enterprise and interactive systems run similar types of jobs which are composed of number transaction-type requests.

Definition 5. *A webJob is defined as <arrivalTime, numberOfRequests>,*

where:

- *arrivalTime* is the arrival time of the job,

- *numberOfRequests* is the number of requests within that job.

We assume that enterprise applications can run on varying number of compute nodes and can expand on number of nodes depending on workload and SLA. We assume that there are a maximum and minimum number of compute nodes that can be used to run the enterprise application; these may vary dynamically depending on the application's workload and SLA requirements. Note that unlike an HPC system, each enterprise application has its own work. Enterprise applications may differ depending on how computationally intensive their workload is. To model this feature of workload, we use *computeIntensity* parameter. As an example of "compute intensity", a computeIntensity = *(1000,1)* means that the application will consume one compute node, i.e., fully utilize (at its primary frequency level or MIPS value) in order to execute 1000 requests.

Definition 6. *An enterprise application is defined as < startTime, duration, computeNodes, computeIntensity, workload, SLA, terminationTime >*

where:

- *startTime* is the time to start running the application.
- *duration* is the length of time that the application is set to run; we assume that if *duration* is 0, then the application is to run indefinitely.
- *compute Nodes= <min, max>*, where *min* is the minimum number of compute nodes needed to run this application and *max* is the maximum number that can be used. We assume that the management system can dynamically increase or decrease the compute nodes it uses for an enterprise application; this is done through the resource manager (see below). If *min = max*, then the application makes use of a fixed number of compute nodes and is assumed not able to change.
- *computeIntensity = <maxNumberOfRequests, numberofBasicNode>*, which defines the CPU intensity of the workload of this application. The *maxNumberOfRequests* and the *numberofBasicNode* are used for performance

modeling where *maxNumberOfRequests* defines the number of requests that fully utilize a n*umberofBasicNode* of basic computes node. In our terminology a basic compute node refers to a server whose computational power has been benchmarked. We prorate the computing power of other servers with this basic node according to their CPU power comparison.

- *workload = { w_1, w_2 , $w_{3,...}$ }* , where w_i is a *webJob*.
- *SLA = <timeThreshold, percentage, EpochTime>* if the *percentage* of *webJobs* requests completed within a *EpochTime* has response time greater than the *timeThreshold*, then an SLA violation will be reported.
- *terminationTime:* it could be 0 if there is no expected time to end, else it is expected duration of running application.

Therefore, an *Enterprise Workload* is a number of enterprise applications, in which they start at their own start times and their workload may have different arrival times. Each application has its duration.

Definition 7. *An Enterprise Workload is defined as < app_1, app_2, app_3, ..., app_n ,..., >*

Where

app_i: is an *Enterprise application.*

An interactive system has the same type of workload as an enterprise system (transaction type workload). Unlike an Enterprise System, where the applications to run are "defined", an Interactive System has a workload which describes the arrival time, duration and characteristics of each of the applications that are to be executed on the system (stream of applications).

Definition 8. *An Interactive Application is defined as <startTime, duration, computeIntensivity, nodes, appWorkload, SLA >*

where:

- *startTime* is the time to start running the application.

- *duration* is the expected length of execution of the application.
- *computeIntensity* = < *maxNumberOfRequests* , *numberofBasicNode* > is the maximum number of requests that fully utilize *numberofBasicNode* of the basic compute node.
- *nodes*= <*min* , *max*> where *min* is the minimum number of compute nodes needed to run this application and *max* is the maximum number that can be used. If *min* = *max*, then the application makes use of a fixed number of compute nodes and is assumed not able to change.
- *appWorkload* = $\{w_1, w_2, w_{3,...}\}$, where w_i is a *webJob*.
- *SLA*= <*ERT*, *percentage*,*EpochTime*> *ERT* is the maximum expected response time for a request. If the *percentage* of requests completed within a *EpochTime* has response time greater than the *ERT*, then an SLA violation will be reported.

To define Interactive system we need to define the workload of interactive applications. *Interactive Workload* describes the sequence of interactive application.

Definition 9. *An Interactive Workload is defined as* $<app_1, app_2, app_{3,...,}app_n ...>$ *where* app_i *is an Interactive application.*

4.2.2 Systems

Now that we have defined our types of workload, we can define a system. As mentioned, a system will have its own scheduler and resource allocator and racks to run its workloads.

Definition 10. *A system Sys is defined as <Type, Workload, Queue, ResManagers, Schedulers, Racks>*

where

- *Type* defines the kind of system: *HPC*, *Interactive* or *Enterprise*.

- *Workload* defines the workload of the system: *Workload* ε {*Interactive Workload*, *Enterprise Workload*, *HPC Workload*}. The workload for a type of system is comprised of jobs/applications to run on that system; these are described in the next section.
- *Queue* is the queue holding jobs to be executed by the system.
- *ResManagers*: is a set of system level resource allocation/management modules, each of which embodies a resource management strategy. Each *RM* *ε ResManagers* is in charge of managing and allocating compute resources in the system and there must be at least one. We assume that an *RM* has the following capabilities:
    - *Resource offering:* allocates compute resources to the applications,
    - *Resource discovery and monitoring*: handles requests from the management system to increase or decrease the computational nodes assigned to an application, assuming that the application allows it (*min < max)*,
    - *Resource selection:* decides on the compute nodes, chassis, and racks that are to become idle (sleep) or make active depending requests from the management system.
    - *Informing management system:* informing the management system about the arrival of a new job.

  Hence, we assume that the *RM* active in each system interacts with the management system. We assume that one or more resource managers are defined by the administrator at the initialization phase of the system. A system may have more than one resource manager, though only one can be active at any time. The management system can change which resource manager is active in a system. Each system has its own resource management strategies.
- *Schedulers* is a set of schedulers, *Sched* ∈ *Schedulers*, where *Sched* implements a scheduling algorithm used for choosing applications or jobs to run, typically those that have been queued, i.e., in *Queue*, and are waiting to execute. We assume that a system can have more than one scheduler and the

management system can change which scheduler to use. Again, only one scheduler can be active at a time.

- *Racks* $\{R_1, R_2, \ldots, R_k\}$ is the list of racks that comprise the system.

A resource manager is associated with each system. We assume that it runs on one of the compute nodes in the system. Upon arrival/departure of any application/job, the system invokes its associated resource allocation algorithm. The primary tasks that a resource manager, and in general any resource allocation algorithm faces, can be divided into four categories: resource modeling and description, resource offering and treatment, resource discovery and monitoring, and resource selection. A resource allocation algorithm must take into consideration the application's requirements e.g. its SLA requirements. Having this in mind, we can expect that resource manager is given some description or characteristics of the application in order to allocate servers to it; we elaborate on this in Chapter 6.

We also assume that any resource change in the system associated with any application resident in the system is done through the resource manager. Resource allocation also means allocating/releasing compute nodes, racks, and chassis at the request of the management system. Any application in the system may also ask for resource allocation/release. Figure 4-2 illustrates the resource manager module associated with a system. A resource allocation request from an application is sent to the resource manager, which then allocates a number of compute nodes to the application.

Upon arrival of new application inside a system and resource allocation for that application by resource manager; information regarding new arrival application (see Chapter 6 for more detail) will be sent to the management system. The management system will use this information first to set up an autonomic manager for the new MO and then, if necessary, to configure the manager as an example configuring **SLA sensors** (definition of will be explained in Definition 13.)

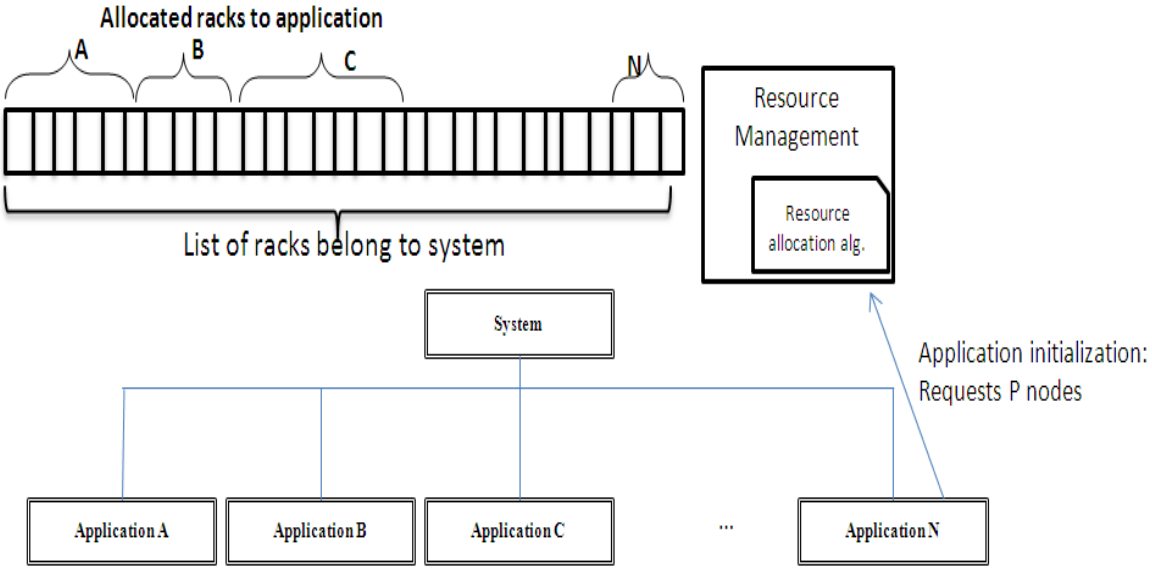

Figure 4-2. Resource management in a system.

The following illustrates two resource allocation algorithms; the first one chooses a node with minimum temperature and the second one chooses a node with minimum thermal effects on the other nodes of the system according to data center thermal model. In doing so, we need to have the list of available compute nodes and their CPU utilizations; the node with minimum CPU utilization is the coolest place.

Algorithm 1. Resource Allocation Algorithm: Choose Coolest Node.

Input parameters: NodeList

1. begin
2. return findMinCPU(NodeList) //Find the node with minimum CPU utilization
3. End

To choose the node with minimum thermal effects on others; we need to have the thermal model of data center (i.e. layout specific model) based on the model we compute thermal effect of all available nodes and then choose the one with minimum thermal effect on the others is the candidate compute node.

Algorithm 2. Resource Allocation Algorithm: Choose Node with Least Thermal Effect.

```
Input parameters: NodeList
1.  begin
2.      for (Node n:NodeList)
3.      begin
4.          therm = thermalEffect(n, thermalModel) // compute thermal effect of
5.            // node n
6.         //update the minEffectNode to point to minimum thermal effect node so far
7.          updateMin(therm, minEffectNode, n)
8.      end
9.      return minEffectNode
10. End
```

There can be other types of resource allocation algorithms which the administrator can define with definition of a system.

We assume that resource managers in any of the data center systems inform the management system about any new application (job) object in the system. In doing so, a resource manager needs to have an extra service besides its core task (i.e. resource allocation) in order to inform the management system about the arrival of new object or even departure of an object.

A scheduler is also assumed to be associated with each system. In our model, we assume that any job scheduling in the system is done through the scheduler. The scheduler decides on the job in queue of the system that is to run next. Figure 4-3 illustrates the scheduling module within a system.

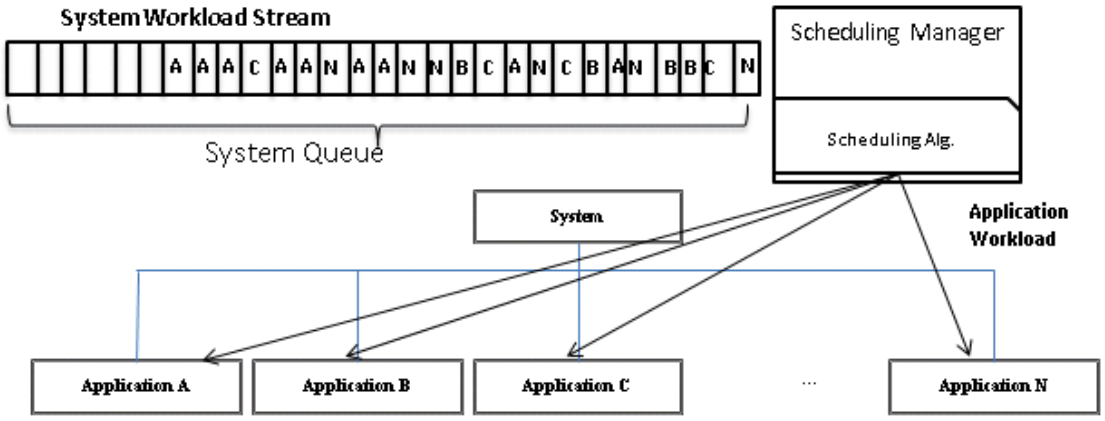

Figure 4-3. Scheduling in a System.

There can be a variety of scheduling algorithms (First Come First Served, Short Job First Served) and they can vary from system to system.

4.3. Manageable Objects in a Data Center

Given above definitions, we can now define the managed elements in the data center relevant to our model. Each element in the model can be seen as an object, which is *potentially* manageable – not that all have to be managed. In practice, the different kinds of managed objects and classes will depend on the administrator; the management system should be able to let the administrator define his/her own managed object classes and corresponding sensors and actuators; the definition and creation of sensors and actuators is beyond the scope of this thesis, but we will assume a number of sensors and actuators in order to help define our management model and for use in our simulator and examples. Before getting to the managed objects definition, we review our definition (*Definition 1*) of a data center.

A Data Center, DC, is defined as DC = <Systems, Racks, ThermalModel>

where:

- *Systems = { Sys_1, Sys , … , Sys_q} is a finite set of systems in the data center;*

- *Racks = {R_1 , R_2 , …, R_k}, where R_i = {r_{i1}, r_{i2} , r_{i2} ,…, r_{ir}};* each R_i is a distinct type of rack, and each rack is a set of chassis r_i= {ch_1, ch_2, ch_3, … ,ch_n} and each chassis ch_j={n_1, n_2, n_3..., n_B} is a set of compute nodes.
- *ThermalModel = <coolerList, RedTemp, ThermalMap>*

Given our definition of a data center, and our definitions of systems, we can define different classes of managed objects. Each category of managed object in a data center would have their own set of sensors and actuators, for example, given two different kinds of racks with different architectures we would define two different classes of managed objects with different sets of sensors and actuators. So, for a data center *DC*, we can define the associated sets of manageable objects. The *Data Center Set of Manageable Objects* (DCSMO or SMO) is defined as follows:

Definition 11. *For a given Data Center, DC, the Set of Manageable Objects MO_{DC} = {MO_{DC}, MO_{Racks}, $MO_{Chasses}$, MO_{Nodes} , $MO_{Systems}$, MO_{Apps}}.*

where:

- MO_{DC} = {mo_{DC} }, mo_{DC} is a managed object representing the entire data center;
- MO_{Racks} = MO_{R1} ∪ MO_{R2} ∪ … ∪ MO_{Rk} , where
    - MO_{Ri} = {mo_{ri1}, …, mo_{rir} }, where R_i = {r_{i1}, r_{i2} , r_{i2} ,…, r_{ir}}, and mo_{rij} is the managed object for rack r_{ij}.
    - We assume that the sensors and actuators are the same for all racks of the same type, i.e., all of the racks in R_i have the same sensors and actuators.

- $MO_{Chasses}$ = MO_{Ch1} ∪ MO_{ch2} ∪ … ∪ MO_{Chk} , where
    - MO_{Chi} = {mo_{chi1}, …, mo_{chir} }, and where R_i = {r_{i1}, r_{i2} , r_{i2} ,…, r_{ir}} and r_{ij} = {ch_{i1}, ch_{i2}, ch_{i3}, … ,ch_{iCi}} and mo_{chij} is the managed object for ch_{ij}.

- We assume that the sensors and actuators are the same for all the chasses in the same kind of rack, i.e., all of the chasses in the racks in R_i have the same sensors and actuators.

- $MO_{Nodes} = MO_{n1} \cup MO_{n2} \cup \ldots \cup MO_{nN}$, where
  - $MO_{ni} = \{mo_{ni1}, \ldots, mo_{nir}\}$, and
  - where $n_{ij} \in ch_t$ for $ch_t \in r_i \in R_i$ and mo_{nij} is the managed object for node n_{ij}.
  - We assume that the sensors and actuators are the same for all compute nodes in racks of the same type, i.e., all of the compute nodes in racks in R_i have the same sensors and actuators.
- $MO_{Systems} = \{mo_{Sys1}, \ldots, mo_{Sysq}\}$; defines a set of manageable objects for all type of systems in the data center. $Sys_i \in Systems = \{Sys_1, Sys_2, \ldots, Sys_q\}$, set of all systems in the data center; we also assume that mo_{Sysi} manages the queue in system *Sysi*.
- $MO_{Apps} = MOA_{Sys1} \cup MOA_{Sys2} \cup \ldots \cup MOA_{Sysv}$, where for each $Sys_i \in Systems = \{Sys_1, Sys, \ldots, Sys_q\}$, MOA_{Sysi} is defined as follows:
  - If Sys_i is an *HPC System*, then $MOA_{Sysi} = \{\}$; that is, there are no managed applications in an HPC system[3].
  - If Sys_i is an *Enterprise System*, with manageable applications $EnterpriseApps_i = \{EntAp_{i1}, \ldots EntAp_{in}, \ldots\}$, then $MOA_{Sysi} = \{mo_{EntApi1} \ldots, mo_{EntApiq}\}$ where $mo_{EntApij}$ is the managed object for application $EntAp_{ij}$. That is, the managed objects associated with an enterprise system are the managed objects for any of the applications it can run that are manageable. .

[3] In the current model, we assume that none of the HPC applications are managed since it is not clear what "managing" an HPC job would entail. This does not have to be the case and this definition can be extended in a fashion similar to Enterprise and Interactive applications

If Sys_i is an *Interactive System* with a set of manageable applications, $InteractiveApps_i = \{IntAp_{i1}, \ldots IntAp_{in}, \ldots\}$, then $MOA_{Sysi} = \{mo_{IntApi1} \ldots, mo_{IntApiq} \}$ where $mo_{IntApij}$ is the managed object for application $IntAp_{ij}$. That is, the managed objects associated with an interactive system are the managed objects for any of the applications it can run that are manageable.

- $MO_{coolers} = \{mocl_1, \ldots, mocl_n \}$; defines a set of manageable objects for all type of coolers in the data center.

The previous definition describes the manageable objects within our data center model. Compute nodes are smallest supported object in the data center model. Considering that the focus of this research is management and energy consumption, in order to model a compute node, beside the computing capabilities of the compute node, i.e. MIPS[4], we need to consider its power features and in any supported alternative CPU frequencies (if any). The following defines the characteristics of a compute node definition in our model.

Definition 12. *A Compute Node object is defined as <MIPS, FullLoadPowerConsumption, ZeroLoadPowerConsumption, StandbyPower>*

where

- *MIPS* = $\{mips_1, mips_2, \ldots, mips_L\}$ are the MIPS levels for each processing frequency level of the CPU; we assume L levels.
- *FullLoadPowerConsumption* = $\{powFul_1, powFul_2, \ldots, powFul_L\}$ is the full load power (i.e. power consumption when CPU is 100% utilized) for each level of CPU frequency.
- *ZeroLoadPowerConsumption* = $\{powZero_1, powZero_2, \ldots, powZero_L\}$ is the zero load power (i.e. power consumption when CPU is 0% utilized) for each level of CPU frequency.

[4] Million Instructions per Second.

- *StandbyPower*: Compute node standby power consumption.

While one can deal with individual managed objects in a management system, it can quickly become overwhelming in handling many different managed objects and, in turn, potentially many different managers. We prefer to consider classes of managed objects, that is, managed objects in the same class have similar characteristics and in terms of management, can be dealt with in a similar fashion, e.g. the same[5] sensors, actuators, autonomic manager. We define a managed object class is as follows:

Definition 13. *A Managed object class MO_Class is defined as <MOs, Events, Sensors, Actuators, MOProperties>*

where:

- *MOs* is a set of managed objects.
- *Events = {$evn_1, evn_2, ..., evn_n$}* is a set of events for the managed object class that can be raised by each of the managed objects in *MOs*.
- *Sensors = {$sen_1, sen_2, ...,sen_n$}* is a list of sensors for the managed object class that are available in each of the managed objects in *MOs*.
- *Actuators = {$act_1, act_2, ..., act_m$}* is a list of actuators (list of functions) for the managed object class that are available in each of the managed objects in *MOs*.
- *MOProperties*: is a finite list of tuples <*Paramameter, Value*>

  where:

  - *Parameter*: is the name of a property or parameter
  - *Value*: is an associated value of the parameter which can be changed or set.

[5] "Same" in this context means that identical properties and behaviors, not the same executable; e,g, to objects may have the "same" manager where each has its own executable from the same image and so specifics to each object may be different.

The above is a general definition of a managed object class.

Our definition of each type of application is included specification of service level agreement (SLA). We assume that for each type of application (MO class); there are sensors that provide an evaluation of the SLA for that application. To evaluate SLA violations we define parameters that are used to determine whether there is an SLA violation or not. The parameters for each type of application are:

- HPC: duration, arrival time, termination time: the SLA for an HPC job is violated when duration plus arrival time of that job is greater than its termination time.
- Enterprise: timeThreshold, percentage, epochTime: An SLA violation happens for an enterprise application when the *percentage* percent of *webJobs* of the application workload response time exceeds the *timeThreshold* expected response time in last epochTime.
- Interactive: expected response time (ERT), percentage, epochTime: the SLA for an interactive application is violated if for the percentage percent of jobs the response time exceed the ERT in last epochTime.

Considering our model of data center and the classes of manageable objects, we can define a set of classes associated with each set of manageable objects, for the data center. Following definition defines the set of managed object classes.

Definition 14. *For a given Data Center, DC, and Set of Manageable Objects* $MO_{DC} = \{MO_{DC}, MO_{Racks}, MO_{Chasses}, MO_{Nodes}, MO_{Systems}, MO_{Apps}, MO_{coolers}\}$*, the Set of Managed Object Classes,* $MOClasses_{DC}$*, of DC is defined as* $= CMO_{DC} \cup CMO_{Racks} \cup CMO_{Chasses} \cup CMO_{nodes} \cup CMO_{Systems} \cup CMO_{Apps} \cup CMO_{coolers}$.

where:

- $CMO_{DC} = \langle MO_{DC}, Event_{DC}, Sensor_{DC}, Actuator_{DC}, Properties_{DC} \rangle$ is a class representing data center objects; where $Sensor_{DC}$, $Actuator_{DC}$ and $Properties_{DC}$ are, respectively, the set of sensors, actuators and properties associated with a managed data center.

- $CMO_{Racks} = CMO_{R1} \cup CMO_{R2} \cup \ldots \cup CMO_{Rk}$ where $CMO_{Ri} = <MO_{Ri}, Event_{Ri}, Sensor_{Ri}, Actuator_{Ri}, Properties_{Ri}>$ is the class representing $MO_{Ri} = \{mo_{ri1}, \ldots, mo_{rir}\}$, and where $MO_{Racks} = MO_{R1} \cup MO_{R2} \cup \ldots \cup MO_{Rk}$. That is, each set of managed objects representing the same kinds of racks are instances of the same class,
- $CMO_{Chasses} = CMO_{Ch1} \cup CMO_{ch2} \cup \ldots \cup CMO_{Chk}\}$ where $CMO_{Chi} = <MO_{Chi}, Event_{Chi}, Sensor_{Chi}, Actuator_{Chi}, Properties_{Chi}>$ is the class representing $MO_{Chi} = \{mo_{chi1}, \ldots, mo_{chir}\}$ in which $MO_{chi} \in MO_{Chasses}$.
- $CMO_{Nodes} = CMO_{n1} \cup CMO_{n2} \cup \ldots \cup CMO_{nN}$ where $CMO_{ni} = <MO_{ni}, Event_{ni}, Sensor_{ni}, Actuator_{ni}, Properties_{ni}>$ is the class representing $MO_{ni} = \{mo_{ni1}, \ldots, mo_{nir}\}$ in which $MO_{ni} \in MO_{Nodes}$.
- $CMO_{Systems} = CMO_{Sys1} \cup CMO_{Sys2} \cup \ldots \cup CMO_{Sysv}$ where $CMO_{Sysi} = <MO_{Sysi}, Event_{Sysi}, Sensor_{Sysi}, Actuator_{Sysi}, Properties_{Sysi}>$ is the class representing $mo_{Sysi} \in MO_{Systems} = \{mo_{Sys1}, \ldots, mo_{Sysq}\}$. We assume that there is a class for each of the managed objects representing each of the systems, i.e., that each system corresponds to a single class. This can be broadened to where there are fewer classes and multiple systems have managed objects in the same class.
- $CMO_{Apps} = CMO_{App1} \cup CMO_{App2} \cup \ldots \cup CMO_{Appz}$ where $CMO_{Appi} = <MO_{Appi}, Event_{Appi}, Sensor_{Appi}, Actuator_{Appi}, Properties_{Appi}>$ is the class representing MO_{Appi} where MO_{Appi} is a subset of $MO_{Apps} = MOA_{Sys1} \cup MOA_{Sys2} \cup \ldots \cup MOA_{Sysv}$, which are the managed objects for managed applications in each of the systems (Sys_i). For $MO_{App1}, \ldots, MO_{Appy}$ we have $MO_{Appi} \cap MO_{Appi} = \emptyset$, that is, the managed object representing a managed application is from a single class and no managed application objects are from more than one class. Note as well, that this definition does not require all applications to be managed.
- $CMO_{coolers} = CMO_{cl1} \cup CMO_{cl2} \cup \ldots \cup CMO_{clk}$ where $CMO_{cli} = <MO_{cli}, Event_{cli}, Sensor_{cli}, Actuator_{cli}, Properties_{cli}>$ is the class representing $MO_{cli} = \{mo_{cl1}, \ldots, mo_{clr}\}$, and where $MO_{clooers} = MO_{cl1} \cup MO_{cl2} \cup \ldots \cup MO_{clk}$. That

is, each set of managed objects representing the same kinds of cooler are instances of the same class.

Notation: We will often refer to an object or component of the data center as belonging to a class rather than specifically referring to its corresponding managed object. As well, for a given data center DC we will let $Events_{DC}$ denote the set of events from all classes of managed objects in DC, let $Sensors_{DC}$ denote the set of sensors from all classes of managed objects in DC and let $Actuators_{DC}$ denote the set of actuators from all classes of managed objects in DC.

The classes of managed objects are important since in our management model we will assume the existence of an autonomic manager associated with each class; more precisely, one that can be instantiated with the details of a managed object from that class. This means that rather than a multitude of autonomic managers, e.g. for each managed object, we have classes and managers for groups of objects. For example, a particular kind of virtual machine would be associated with a class and on deployment an autonomic manager with the necessary sensors, actuators and properties of that virtual machine would be deployed. Other virtual machines of the same type would also be able to have a manager formed similarly.

4.4. Summary

In this Chapter we have provided our abstract model of a data center and corresponding to that model we have related managed objects and for each class of managed object we have defined a corresponding class with the definition of a finite set of sensors and actuators. By having this model of data center and classes of managed objects, we will be able to define our management model which we do so in the next Chapter.

Chapter 5

Management System Model

In Chapter 4, an abstract model of data center was presented. Our focus in this Chapter is on the development of a framework for specifying policy based autonomic management systems for data centers that is based on the model introduced in Chapter 4. In this Chapter, we introduce our management framework; we first discuss the principles of the management system and then define our model.

5.1. Management System Principles

We assume that the management system can operate in any data center, with various management levels and different locations of autonomic managers (AMs). The management system is policy based; this means that each manager has its own set of management policies governing associated managed objects and has policies for dealing with the rest of management system.

We assume that policies are defined as **Event-Condition-Action (ECA)** policies and the set of policies associated with an AM are referred to as its ***Policy Profile***; an AM may have multiple policy profiles associated with it, but only one is active at a time**.** A set of policies can be altered by an administrator or could change based on a particular system situation.

When a system starts or when applications or jobs are deployed, autonomic managers may also be initiated or existing autonomic managers could be assigned to manage that application. The conditions on which systems and applications are managed is part of the configuration information provided by the administrator to the management system. The models in the previous Chapter and in this Chapter provide the core of this configuration information required in our management system. This Chapter specifically focuses on the management system model.

When autonomic managers (AMs) need to be deployed, the management system must ensure a number of things. Each AM in the management system first needs to be initialized, including getting its associated set of active policies and knowing the other AMs that it needs to communicate with, e.g. in a hierarchical organization this could be the parent. Once an AM is initialized, it can activate its sensors and actuators, establish its communications with other AMs and start to execute based on the MAPE loop. The management system must also terminate AMs when applications end. It must also ensure that communication among the remaining AMs takes place according to the overarching AM topology or identify a problem. In Chapter 6, we describe algorithms to support these activities consistent with our models.

5.2. Definition of a Managed Data center

A managed data center is a data center (as defined in Chapter 4) which has a management system associated with it. To form the management system, first we need to know which objects need to have a manager. From Chapter 4 we have a definition of a "data center" and the kinds of "managed objects" in the data center. Again, these "managed objects" are assumed to be the kinds of objects that *can be* managed within the scope of that data center model; the objects that are *actually* managed are specified by the administrator and are assumed to be a subset of the "managed objects" defined.

In this section we aim to define the model of our management system. Our first step is to associate a manager class with a class of managed objects.

Definition 15. *For a given Data Center, DC, and* $MOClasses_{DC}$ *of* $DC = CMO_{DC} \cup CMO_{Racks} \cup CMO_{Chasses} \cup CMO_{nodes} \cup CMO_{Systems} \cup CMO_{Apps} \cup CMO_{coolers}$*, we define the set of Autonomic Manager Classes of DC,* $AMClasses_{DC}$*, to be =* $\{Am_i \mid Am_i$ *is a class of autonomic managers associated with* CMO_i *where* $cmo_i \varepsilon MOClasses_{DC}\}$.

Abstractly, we associate a class of autonomic managers with each class of managed objects. Instantiating a managed object would (assuming that the object was to be

managed) result in the instantiation of an autonomic manager. Pragmatically, one can think of a class of managers to be a parameterized autonomic manager that is assigned specific details about an object to manage when it is initialized. As well, the management system may or may not contain managers for all MO classes; this depends on the administrator and the architecture of the management system.

Managers in the management system can be configured to interact with one another. We could leave the administrator to explicitly define the communication among managers, but this is intractable in a data center where there may be hundreds of managers. Further, we would like to automate this as much as possible. This is the notion of an *autonomic manager topology*. For a specific set of managers their communication represents a single topology. As managers change, so does the topology. Hence, we need a general model of topologies for a data center; we do this in terms of manager classes.

A *management pair* in the management system has two manager classes and *priority.* The pair of classes means that managers in those classes can communicate with each other. The priority determines which Autonomic Manager Class (AM class) in the pair is a privileged, if there is one. AMs of the privileged class can be thought of as "managing" AMs of the other AM class. If there is no privileged AM class, then AMs in the two classes are essentially peers. A privileged AM can, in essence, take actions to influence the behavior of an AM in the other class, including, for example, changing the policies of the subordinate AM. These actions are done through communications between the managers; we will elaborate on this later.

Definition 16. *For a given Data Center, DC, the Set of Managed Object Classes, $MOClasses_{DC}$, and a Set of Autonomic Manager Classes, $AMClasses_{DC}$, a Management Pair set, AMP, is defined as:*

$$AMP = \{(A_i, A_j, \text{priority}) \mid (A_i \text{ and } A_j \in AMClasses_{DC} \text{ and priority} \in \{0,1\}\}$$

For example, if (AM_1, AM_2, *priority*) is a management pair, then if the *priority=1* that means that AMs of AM_1 class are privileged can change the settings or behavior of AMs of AM_2.

A management topology is then defined as a set of manager classes and the management pair set corresponds to vertexes and edges in a graph.

Definition 17. *For a given Data Center, DC, Set of Managed Object Classes $MOClasses_{DC}$ and a Set of Autonomic Manager Classes, $AMClasses_{DC}$, a Management Topology T_{DC} for DC is a graph G of <V, E>*

Where:

- $V= \{v_i | v_i \in AMClasses_{DC} \}$.
- $E = \{e | e = \{(A_i, A_j, \text{priority}) | A_i \text{ and } A_j \in AMClasses_{DC} \}\}$

A management topology actually defines the relationship between different management classes attached to managed object classes. In specifying the topology, one specifies constraints on the relationships that can occur between different managers. In practice, this topology will be instantiated during data center administration and operations resulting in a graph connecting managers that is consistent with this topology.

Having defined a management topology, data center, classes of managed objects, and classes of managers, we can define a *managed data center*, as follows:

Definition 18. *A managed data center MDC is defined as <DC, T>*

Where:

- *DC = <Systems, Racks, ThermalModel>, is a data center with Set of Managed Object Classes, $MOClasses_{DC}$, Set of Autonomic Manager Classes, $AMClasses_{DC}$,*
- T_{DC} is a management topology for DC.

The definition of a Managed Data Center defines the management framework. In practice, depending on the administrator, within that framework not all managed objects need to be considered. That is, one can think of the framework as defining ***what can*** be managed and relationships among management elements, while ***what is*** actually managed can be dictated by the administrator. Abstraction level in the management system is illustrated in Figure 5-1. At the first level, we have set of possible manageable objects; here we assume that these are objects which are to be managed as well as autonomic managers that have been or could be defined. We assume that these are grouped into classes (the second level). This allows objects with similar management properties, e.g. sensors, actuators, to have a single definition and that properties of a specific object can be instantiated. Third level is the autonomic management class level; each AM class corresponds to one or more MO classes. This allows an autonomic manager to be instantiated, i.e., its properties specified, when an object is instantiated. Allowable connections among autonomic managers are defined as autonomic manager pairs at the next level. The connections between classes of autonomic managers define the autonomic manager topology of the management system.

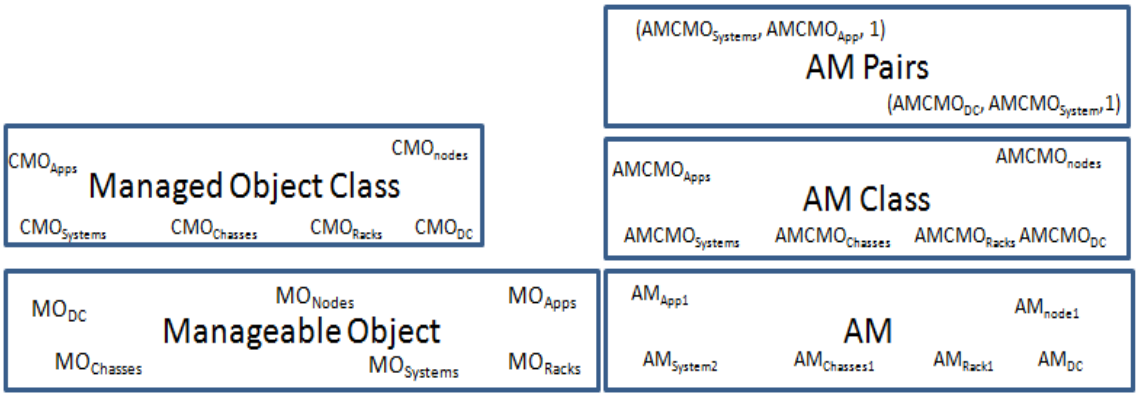

Figure 5-1. Abstraction Level in the Management System.

5.3. Policy based management

Our aim is to have a policy-based management system. In a policy-based management system, decisions regarding actions are expressed as a policy. In this section we define policies associated with a managed data center.

For our definitions of policies, we need relational and conditional expressions. For this thesis we adopt the general notion of these expressions as in many programming languages. We assume that a *relational expression* is a Boolean expression of the form op_1 or of the form op_1 *relop* op_2, where op_i is an expression that evaluates to a common value (integer, float, Boolean, etc.) and *relop* is a common relational operator (e.g., =, <, >, etc.); we assume that a single expression without a relational operator evaluates to a Boolean value. A *conditional expression* is then a finite set of relational expressions, which we interpret as a conjunction of those relational expressions. More general forms of conditional expressions can be accommodated, but this suffices for our current model.

As described earlier, we assume that we have Event-Condition-Action policies. We define a policy to be:

Definition 19*. For a given Data Center, DC, Set of Managed Object Classes* $MOClasses_{DC}$ *and Set of Autonomic Manager Classes,* $AMClasses_{DC,}$ *a policy Plcy is defined as* $< E, C, A >$

where:

- *E*: is a list of events, $<event_1, \ldots, event_n>$, where $event_i \, \varepsilon \, Events_{DC}$.
- *C*: is a condition expression $\{c_1, \ldots, c_m\}$, where if $c_i = op_1$ or $c_i = op_1$ *relop* op_2, then $op_1, op_2 \in Sensors_{DC}$.
- *A*: is a list of actions, $<act_1, \ldots, act_p>$, where $act_i \in Actuators_{DC}$.

While we can represent a policy as a triple (<E,C,A>), policies are often written in some stylized language, e.g. Ponder [47]. This provides a useful human-oriented form for

expression (and can often be used to provide other additional information). We adopt a stylized form for expression of policies in this text to aid readability:

Given a policy <E,C,A> = <<$event_1$, …, $event_n$>, {c_1, …,c_m}, <act_1, …, act_p>>, we can represent it as:

On Event: $event_1$ or …or $event_n$
If (c_1 & … & c_m)
Begin
act_1, …, act_p
End

There may be some precedence in invoking actions; Table 5-1 illustrates number of ponder[6] operator assumed in this thesis, and their explanations, e.g. operators for defining concurrent actions and events evaluation.

Table 5-1. Examples of Ponder Operators

Operator	Explanation
a1 -> a2	a2 must follow a1. If any of the actions fail or is not allowed by a refrain policy, the execution stops.
a1 | a2	a1 is performed. If it fails or is not allowed by a refrain policy, a2 is performed. If a1 succeeds, execution stops. (action operation)
e1 | e2	Occurs when either e1 or e2 occurs irrespective of their order (event operation)

We refer to a finite set of policies as a *policy profile*. The notion of a policy profile is a useful management concept in the context of a management system in that frequently a number of policies are group together to achieve a broader goal.

In this context, given a managed data center, there are policies that can be formed to manage the objects in the data center. The policies are dependent on the managed classes

[6] A description language for specifying policies is not the focus of this thesis. We write policies in pseudo code based on Ponder.

associated with that data center and, of course, can be altered by the administrator and, as we shall show, by other autonomic managers. We refer to the management system associated with a managed data center as a ***policy based management system.***

5.4. Communication in the Management Model

Our management system assumes that multiple autonomic managers cooperate by exchanging messages. In our definition of a managed data center, we defined a topology of managers where the topology defines the AMs that can communicate. We also introduced the notion of "privilege" – that one manager has privileges over another. This is realized in the kinds of messages that can be exchanged and the implications of those messages. Each manager in the management system can communicate with a number of other AMs as defined in a pair relationship among AM Classes (Definition 16.). When an AM is initialized (see Chapter 6, section 6.4) it is given a list of its "peers", that is the list of AMs that it can communicate with, including those that have privileges over it, those that it has privileges over, and others (peers).

How does an AM know when it is time to send a message? The answer is "sending messages should be specified in the policy." Assume that an AM needs to change profile policy of another AM, this change should be specified in the action part of a policy. In this case, the action is "Send message" with a new profile policy Id. "Send message" is assumed one of the actions that are part of an AM class.

While there are many possible kinds of messages that can be exchanged between managers, we define a core set for our work; the types of messages can always be extended.

Table 5-2. Management System Core Messages Set

Message taxonomy	Type of message	Description
AM life cycle	Destroy/Initialize	A new AM sends an initialize message to AMs that is in any relationship with itself (privileged or not privileged). An AM that has finished sends a message to

		AMs is in relationship with itself (privileged or not privileged) to let them know that it is leaving the management system.
Update information	ReqForHeartbeat	An AM asks for heartbeat values from another AM.
	UpdateHeartbeat	An AM updates its heartbeat values and sends them to a requesting AM.
Policy change	ChangeProfilePolicy	A privileged AM updates the policy set of another AM

We have envisioned a basic set of messages for management system operations; details are presented as follows:

- <*Initialize, AMId, PrivilegedCode*> A new AM sends an initialize message to any AM in its neighbor list announcing that it is active (born) along with its identifier, *AMId* . An AM has three different types of relations with its neighbors; it can be their parent, their child, or their peer. The type of relationship is determined by the last parameter of the message, *PrivilegedCode*. *PrivilegedCode* can be IAMCHILD, IAMPARENT, and IAMPEER. The receiver *AM* will register the AM with that specified *AMId* in its internal registry.
- <*Destroy, AMId*>: When a managed element finishes, for instance when an application is done, the corresponding AM needs to be removed from the management system. For instance, for a user that uses number of compute nodes to run an application, when that application has completed, the AM managing that application needs to be removed. When the AM determines that its application has terminated, the AM will send a message to its neighbor list and afterward will terminate. The receiver AMs will delete the AM *(AMId*) from their neighbor list. This message will be sent to all AMs in the neighbor list of the AM.

- <*ReqForHeartbeat, info, AMId*>: When an AM wants to get information about a neighbore.g., sensor information, the AM will send this message to that specific

neighbor with *AMId*. Such information, for example, may be needed to evaluate a condition in a policy.

- <*UpdateHeartbeat, AMId, info, eventFlag*>: In response to a *ReqForHeartbeat* message, an AM will send back the sensor values that are specified in the request message to its peer encapsulated in the *info* part of the message. Obviously, the *info* part is AM specific and may be different for each AM. The last parameter *eventFlag* is a flag to specify that if this message is a response to a ReqForHeartbeat message or the AM wants to trigger an event inside the other AM. If *eventFlag* has the value "**EventCode" then that** means that the sender AM wants to notify the receiver AM of its need and triggers an event inside the receiver AM otherwise the intention of this message is just updating heartbeat values. In this way message passing is used to trigger an event outside of an AM in our management system. In case of triggering an event, the receiver AM should have this event in its event list and also have a related policy that will be triggered by this event. Otherwise, we assume that the event the triggered event will not have any impact on the receiver AM. In section 6.6 event handling is explained with an example.
- <*ChangePolicyProfile, AMId, profileId*>: When a privileged AM wants to change the active policy profile of a child, it encapsulates identifier of the policy profile and the child *AMId* into a message and sends it to the AM. Upon receiving this message from the parent AM, the AM contacts the policy repository for the policy profile with *profileId* and downloads the corresponding policies. Once the policies are downloaded, the AM will invoke the new policies on the next management cycle. This message can be sent to all AM in the peer list that have privileged relation with the AM and AM is the privileged AM on that relation.

5.5. Summary

In this Chapter, the principles of our management framework have been presented. The management system framework considers each component in the data center as an MO which potentially may have an AM that manages it specifically. The AMs in the data center know their neighbors and communicate with them to share status information and any configuration information regarding operational aspects of managed objects or manager behavioral. The cooperation between managers is done through policies. At any time, the active set of policies for an AM defines the behavior of the manager.

Based on the discussion of collaboration models among agents as discussed in Chapter 2, our management model would be considered a data integration model.

In the next Chapter, we will focus on the algorithms that a management system requires for deployment and operations.

Chapter 6

Management System Deployment and Operation

After defining the management model in previous Chapter, the next step is to define the how to instantiate a specific management system and its operation. This Chapter presents the algorithms that are needed to deploy the management system and to instantiate objects from the management system model.

In a data center there are static managed objects (MOs) (e.g. racks, computers, coolers) and dynamic MOs, e.g. various types of applications that may come and go, so we may have a dynamic number of MOs and respectively, a dynamic number of AMs. Therefore, the management system is first configured to know the classes of static MOs and possible dynamic MOs. Upon arrival of an MO – for example a new application - in the data center environment (assume that the management system needs to have an AM attached to the application), the AM initiation module is invoked for the new AM in order to connect it into the management system.

The administrator defines MOs, their classes, the associated AM classes and their topology before the management system startup. This could be done in a variety of ways – through configuration files and scripts, through some interactive tool, etc. We also assume that any changes to the management system, e.g. specification of a new policy, will be done through some such interface; details of this are beyond the scope of this thesis. Before the entire systems are booted (i.e. started) the administrator have to specify the classes, MOs, managers, AM classes needed prior to starting up the management system.

We assume that there are separate spaces of *computing* and *management* in our model. That is, there are number of compute nodes dedicated to running the management system and the policy repository. So, before any of systems from computing space in the data center come online, the management system compute nodes should be active and running. The management space is represented as one separate system, which has its own resource manager and set of servers, i.e. servers that will be allocated to autonomic

managers and the management system controller (explained below). It is also important to realize that AMs can run as different processes inside a server; therefore, we primarily need servers in the management space for running the core of the management system. To decrease the overhead of communication between managers and MOs, managers need to be located close to the MOs. In this case managers could be running on one of the same systems as the MOs. To figure out where is good place to place the manager considering the communication load (communication with other managers and its associated MOs) is the key, but is beyond the scope of this thesis.

The primary algorithms for deploying and maintaining the management system are:

- Initializing the management system; i.e., start-up.
- Setting up an AM when a new managed object arrives and determining how it is connected within the management topology.
- Defining the management loop in an AM.
- Handling messages.
- Termination of an AM.

The management system has one or number of nodes that we shall refer to as the *management system controller;* we refer to the nodes restricted to management services as the *management space*. The controller is located in management space and is up before any of managers in the data center come online. The management system controller is responsible for:

- Starting up the management system based on the administrator configuration scripts;
- Maintaining information necessary for the operation of the management system. i.e. AM table, instance table, topology table and MO table (all explained later in this Chapter).
- Handling the arrival of a new object to manage and AM instantiation.

- Providing the interface between the management system and the administrator. If the administrator aims to make any changes in the configuration of the management system, they would have to log in to the management system and define the new configuration, which would then be processed by the management system controller.

6.1. Management System Instantiation

The administrator must log into the management computing system and specify the details of the management system model for the data center. The specific management system can then be instantiated, initialized and made operational. At this point, the management system is in a state prepared for the start of the systems and applications. Note that this process is likely only done when all systems are to be started. In practice, the management system will be running and as new systems come on-line and others are taken down, changes to the management system would happen through the administrator.

To instantiate an instance from the management system model, the administrator needs to specify:

- The set of MO objects and their associated classes that are to be managed (recall that not all objects included in the framework need to be managed); this would include specifying the events, sensors, actuators associated with each class of managed objects.

- The set of AM classes that correspond to the MO classes whose objects are meant to be managed; this also assumes that an actual manager (i.e. executable) is defined and associated with that class.

- The AM Topology.

- A Policy repository where policies will be stored and managed.

We expect that administrator defines these and the management system takes over setting up the system and maintaining it. In practice, of course, these definitions may have

already been defined and the administrator simply selects classes of objects, etc. that are to be managed. Figure 6-1 illustrates the interaction between the administrator, the management system, the data center, and the policy repository.

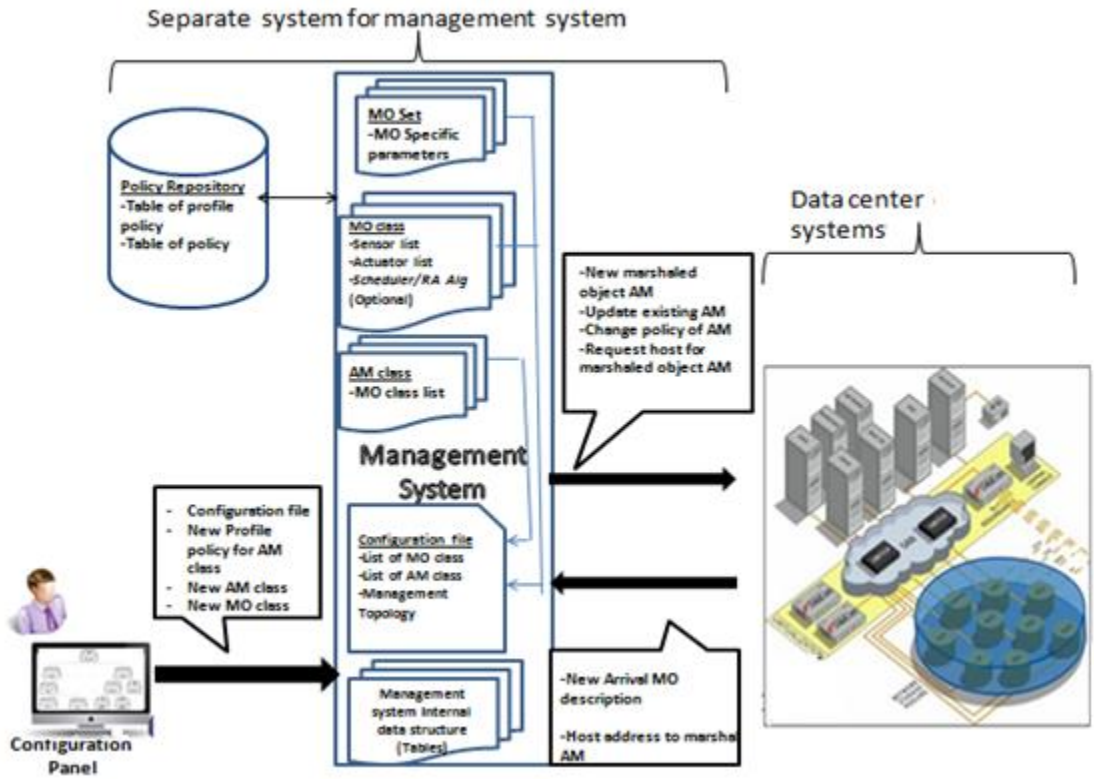

Figure 6-1. Administrator, Management System, and Data Center at Runtime.

As shown, there is a separate system for the management system beside the production computing systems in the data center. Suppose that the administrator interacts with management system through a configuration panel (alternative approaches for interaction are also possible). The administrator describes the classes of managed objects and associated classes of AMs for them. The topology of the AM classes is defined by administrator in a configuration file. The management system reads these configuration files, processes them and puts them in internal tables. These tables will be explained later in this Chapter.

The management system communicates with the policy repository; upon any change in a policy or changes in the policy profile of any managers in the management system. For instance, if the profile policy of a manager needs to be changed, the manager needs to communicate with policy repository to get a new set of policies. Upon the arrival of any new MO to a system in the data center, the resource manager on that system (as the first node to get informed about new arrival MO) must inform the management system controller about the new MO. As explained in Chapter 4, a resource manager (*ResManagers* refer to Definition 10.) in the system is in charge of allocating node/s for new application inside a system. Thus, the resource manager that knows about new MO needs to inform the management system upon arrival of a new application, then the management system will decide whether to dedicate a manager to that or not. How does the resource manager inform the management system about the new MO?

The resource manager on the system needs to send information about new MO e.g. its system. But where does the resource manager get the information about the new MO?

The primary tasks that a resource manager, and in general any resource allocation algorithm faces, can be divided into four categories: resource modeling and description, resource offering and treatment, resource discovery and monitoring, and resource selection. A resource allocation algorithm must take into consideration the application's requirements e.g. its SLA requirements. Having this in mind, we can expect that resource manager is given some description or characteristics of the application in order to allocate servers to it. Such information is very common, e.g. number of compute nodes, memory requirements, etc., and we assume that additional information can be provided. So, after initialization, it can forward this description to the management system. This information is just a description of the new MO and the resource manager has no idea about the new MO's class in the management system. It is the management system's responsibility to figure out the class of managed object for this new MO according to the information it gets from the resource managers. What type of information is needed? For our purposes, we assume that a resource manager will provide:

- *Runtime Details*: Details on the application's runtime requirements, e.g., number of compute nodes requested, memory, expected execution time, compute nodes allocated, primary compute node, etc.

- *Description*: A "description" of the new MO e.g. application, VM. This description will be used to identify the class of MO. For instance for an application running inside an enterprise system this field will be just *application* management system will figure out which class of enterprise application should be used by putting all information coming from resource manager.

- *System Info*: The system that the object will execute on, e.g. system identifier or address, type of system (e.g. HPC). System info will be used to identify the class of MO since we know all MO applications inside the system have the same type of workload.

- *SLA Info*: Its SLA description parameters.

The management system will use this information to first figure out the MO class for this new MO, and second whether the management system needs a manager for it or not. If the management system cannot determine the MO Class of the MO from the provided information, we assume that it can treat the MO as a "non-managed" object (i.e. there is no AM) or can request information from the administrator. In this latter case, the management can accumulate information about new applications and the classes they belong to (presumably, the administrator could also define a new MO Class and AM Class for a new application if required).

If there is to be a manager, then it must determine the neighbors of the MO's manager in the management system (assuming that the new MO is supposed to have an AM). We refer to this information as MOPhysicalInfo (a record of information about the new MO) in our proposed model.

The management system stores the information about managed objects and their classes, managers and their classes, topological constraints among managers, etc. Each MO class

has its set of sensors, actuators and events. As noted, the managed object class configuration file describes the list of managed object classes; the autonomic manager configuration file describes the list of manager classes. The topology configuration file holds AM pairs (AM topology) in the management system.

Algorithm 3 *(MS_StartUp)* illustrates the pseudo code for starting up the management system and configures it based on given configuration files. For static managed objects in the system there is a configuration file (entered by the administrator) which is read and their AMs will be instantiated. Note that AM instantiation for static objects is the same as AM instantiation for dynamic objects; instantiation will be explained later in this Chapter. The management system is then started.

Algorithm 3. Management System Start up.

MS_StartUp:
Input parameters:
1. begin
2. Read Managed Object Classes and Autonomic Manager Classes from configuration files and construct Table 6-1; table of correspondences between the Manager Classes and Managed Object Classes.
3. Read the management system topology configuration file and construct Table 6-3.
4. Read the static managed objects configuration file and deploy associated AMs for the static objects.
5. End

6.2. Policy repository

The proposed management system is *policy-based* which means that each AM has its own set of ECA policies referred to as a *policy profile*. This set can be altered according to the system situation or even a direct change from the data center administrator. We assume the existence of a *policy repository* which holds the policy profiles associated with each of the managed element classes. Figure 6-2 illustrates the relationship among the MO classes, policy profiles and individual policies. Each class of MOs could have multiple different associated policy profiles which could involve different sets of ECA policies. Generally, though, we expect that there would be much overlap, e.g., a policy

for managing an application during a work day would be very similar to the policies for managing it at night or on a weekend except perhaps for some parameters.

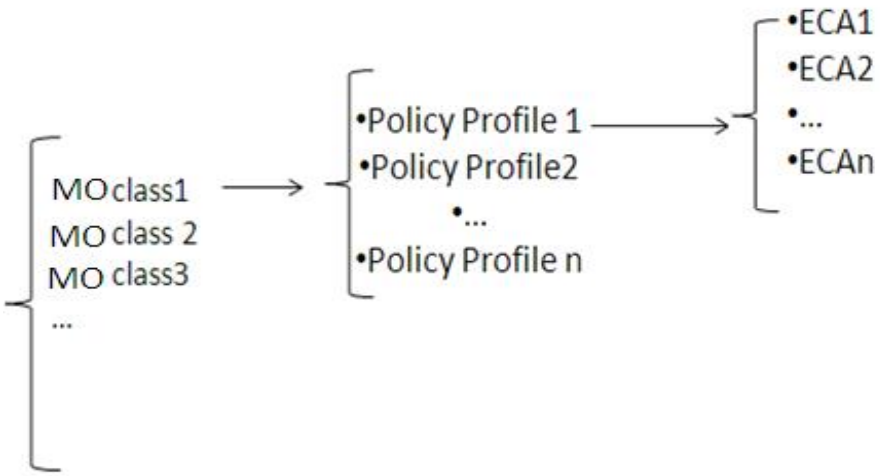

Figure 6-2. Information flow to determine policy.

The *Policy repository* server contains the policies for the AMs in the management system. The policy repository could be a database server, an LDAP server or a Directory Enabled Network (DEN) device. All AMs during their life cycle have access to this server, to read their respective policies. The first time that an AM has access to the *policy repository* is at its initialization, although during its life cycle the AM may access the repository to get updated policies.

6.3. Arrival of a New Managed Object

The arrival of new MO starts from when the resource managers in the system inform the management system the arrival of new object at a system in the data center.[7] The resource manager of the system sends information about the new object (MOPhysicalInfo) to the *management system controller*; which includes information about the type of the object and the system on which the object will run. This information is received by the

[7] Note that there might be number of new object arrival notifications in the management system, i.e., for each of the system; these requests will be queued and processed by the management system.

management system controller which maintains a list of MOs and their associated information. Now, if an AM is needed for the new object, the management system asks its own resource allocator (as earlier explained management system has its own resource allocator) to assign a node for its AM. [8]Afterward, the management system will initialize the AM. This flow of work is shown in Figure 6.3.

Note that the focus of this thesis is on automating management activities, so the administrator specifies MO classes that should have AM and the management system create a manager for that. In our experiment, we focus on having one manager per MO though it is not a requirement for the management system. AMs can manage more than one MO at the time and it can address each of them based on their ids.

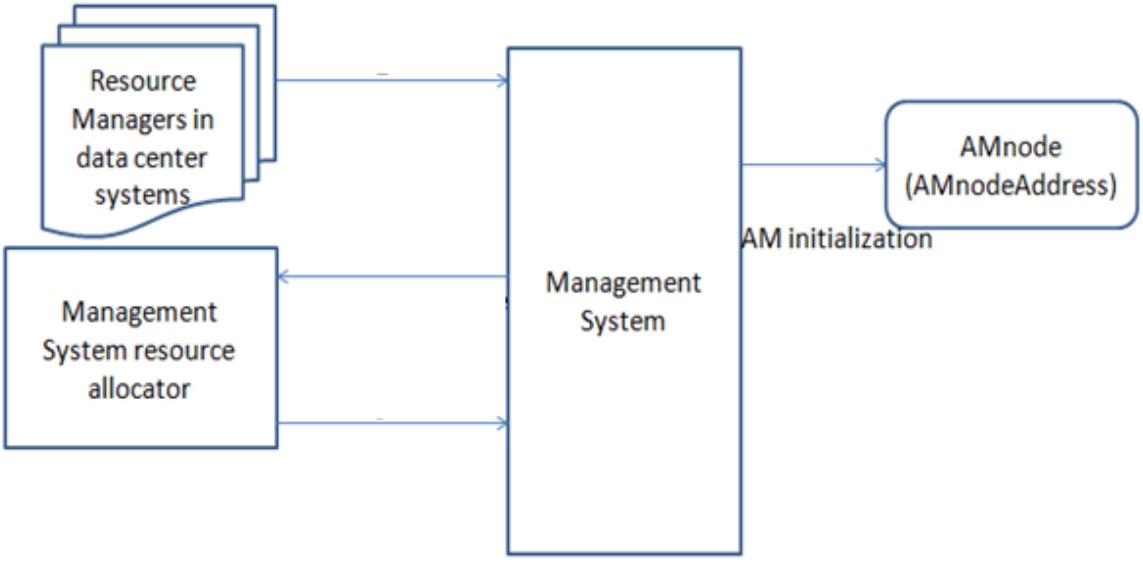

Figure 6-3. Management Work Flow for the Arrival of a New Managed Object.

The management system will associate an AM for the new MO if the administrator had specified AMs for that class of MOs. The administrator needs to specify the MO classes that need to have AMs and a manager object class for that class of MOs.

[8] Considering overhead of communication between a manager and its MO another approach for allocating node to run the manager is asking the system that MO is resident there to allocate a node for the manager. In this case, the manager is not located in the management system's system instead the manager is in system associated its MO.

The management system has internal tables to save information about MOs, AMs and associated classes. Table 6-1 (we name it *AM Table*) exemplifies how the management system stores AM classes and the corresponding AM class needed for instantiating the new AM. The administrator specifies what manager class should be allocated to each MO class and the table is constructed upon startup of the management system (refer to Algorithm 3). The management system refers to this table upon the arrival of a new MO to check whether an AM is needed or not and if so which class of AMs should be instantiated.

Table 6-1. *AM Table*: Correspondences between AM Classes and MO Classes.

MO class List	Associated AM Class
VM	VM_AM
Cluster	clustreAM
Rack	rackAM
Application.Interactive1	InteractiveApp1
Application.Interactive2	InteractiveApp2
Application.Enterprise1	EnterpriseApp
System.HPC	SysHPC
System.Enterprise	SysEnterprise
System.Interactive	SysInteractive
…	…

So far, the deployment algorithm has figured out the necessity of having an AM for the new MO in the management system.

The management system controller maintains update information about what exists in the data center so it will store the information about new MOs. The *MO Table* (refer to Table 6-2) is another internal table in the management system that for each class of MOs has a list of all available MO objects with their corresponding received MOPhysicalInfo (received from corresponding resource managers) regardless of having manager for it or not. In other word, this table actually shows the object of each class of MOs. Management system will figure out the class of managed object according to the

information in MOPhysicalInfo. MOPhysicalInfo is the same for each MO class with different parameters.

This table gets updated upon arrival of new MO from a MO class.

Table 6-2. *MO Table*: MO Classes and their Information.

MO Class	MOPhysicalInfo List
App	App1Info, App2Info,..
System-HPC	Sys1Info, Sys2Info
…	…

Now, the AM initialization algorithm needs to get the list of AMs to be connected to the new AM. To do this, the management system needs to refer to the topology model (illustrated in previous Chapters) to figure out the AMs associated with the new AM. Table 6-3 (i.e. named the *Topology Table*) is formed from the topology specification provided by the administrator. For instance, according to this table, an AM attached to an application of class "App" needs to connect to AMs attached to other applications of class "App" with the same privilege and also to the AM attached to the system class "System", where that AM is privileged over an AM from class "App". Also, the AM connected to an HPC-system needs to be connected to just the data center AM. Table 6-3 shows the connection between classes of AMs, to deploy this connection we need to know the instance of each class.

Table 6-3. *Topology Table*: AM Class Pairs.

AM class	AM Class Pair List
App	(System,1) , (App,0)
System-HPC	(Data Center,1)
….	….

Another necessary mapping to completely describe the runtime picture of the management system, is a mapping from any AM class to a list of all instances from that AM class (note that this is table gets updated at run time by the management system). The Management System refers to Table 6-4 (i.e. called the *Instance Table*) to get all of

the neighbor instances and records their IDs in the neighbor list of the new AM. Afterwards, the new AM, in its initialization module, will send messages to these AM instances to inform them of its existence.

Table 6-4. *Instance Table*: AM Class and their Instances.

AM Class	AM Instance ID
App	AMApp1,AMApp2,..
System-HPC	AMSys1, AMSys2
…	…

Table 6-5 contains the list AM IDs and its associated MOs. Given the information about an MO, by referring to Table 6-5 the ID of its corresponding manager can be determined. Note that information about the managed object is all the information that we have from its MOPhysicalInfo.

Table 6-5. AM ID and Corresponding MO information

AM Instance ID	MO
AMApp1	App1Info
AMSys1	Sys1Info
AMApp2	App2Info
…	…

6.4. Autonomic Manager Set-up and Initialization for a new Managed Object

How does the management system get informed about a new object, e.g. a new started application? As described, the resource manager in each system is first to know about the arrival of a new object in the system. After resource allocation to the new object, the resource manager has information about the new object which is then shared with the management system. The resource manager in each system sends information about new object arrival to the management system (this information is named as MOPhysicalInfo in Algorithm 4).

Upon arrival of new MO, the management system updates its internal tables and prepares required information for initialization of the new manager. (Refer to Algorithm 4. *AM_Incarnation*) Algorithm 4 specifies the steps that the management system must take to deal with the new MO.

The management system can definitely have multiple MOs per AM in that case upon arrival of the MO in Algorithm 4 the management system can be extended to pass the id of the AM (i.e. already in the system); then the management system puts the current new MO in the list of MO of that AM. In our experiments and algorithms we are assuming that there is one MO per AM.

Algorithm 4; it first obtains the related AM class for the new MO (getNewAM) according to information in MOPhysicalInfo (i.e. type of new MO, MO id). Then management controller checks the topology to obtain neighbor AM objects for the MO's AM. (In the topology definition we define classes but internal tables contain objects of each class). AMs that are neighbors of the new AM are captured in lists: called AM_Peer_List, AM-Children_List, and AM_Parent. Afterwards, the management controller asks the management system resource allocation module for a new node on which to run the new AM, i.e. AMnodeIP in the algorithm. Now that we have our AM node ready to use, it is time to update internal tables, i.e. the *MO Table* and *Instance Table*.

Algorithm 4. AM Incarnation for a New Managed Object

AM_Incarnation
Input parameters: MOPhysicalInfo

```
1.  begin
2.  if isAMRequested(MOPhysicalInfo) = true
3.  begin
4.      //configure parameters of the AMclass
5.      newAMclass = getNewAM(MOPhysicalInfo)
6.      AM_Peer_List = getPeerFromTopology(MOPhysicalInfo)
7.      AM_Parent = getParentFromTopology(MOPhysicalInfo)
8.      AM_Children_List= getChildrenListFromTopology(MOPhysicalInfo)
9.      //RA allocates a compute node and instantiate a new AM of AMclass on
10.       the AMnode
11.     AMnodeIP=resourceAllocation(newAMclass);
12. //Update internal tables (refer to Table 6-4) update the MO table and instance
    table with the newArrival AM information
```

13. updateTables()
14. end
15. end

The management system internal tables are updated and a new manager at address AMnodeIP is ready to be initialized as an AM for the new MO. The next step is to configure the new AM.

After AM incarnation; the AM needs to be configured to know its associated MO configuration parameters. There is specific information for all MO instances from the specific Class that the MO belongs to and this information needs to be passed to its AM as input parameters or configuration parameters; this information is passed via a vector *configVector* or the form <(param$_1$, value$_1$), … (param$_n$,value$_n$)>, i.e., a vector of pairs of parameter names and values.

The most important parameters of a manager that need to initialize are MO id and SLA sensors related parameters. As explained in the model, each type of application has its own definition of SLA sensors. Now the question is where does this value comes from? And when is this information passed to the associated application AM?

SLA sensor parameters are related to the MO specifications and at MO instantiation time in the system this information gets determined. We assumed that resource manager in each system sends information about SLA parameters of the new MO to the management system as part of MOPhysicalInfo. If this information is not provided, then we assume that there may be default values defined as part of the MO class; if not, then this application has no SLA requirements. This information will be passed to the AM initialization algorithm via *configVector*. So for each type of interactive and enterprise workload the SLA sensor related parameters are as follows:

- Interactive: (*ERT, percentage, EpochTime*)
- Enterprise: (*timeThreshold , percentage, EpochTime*)

Note that MOPhysicalInfo is MO dependent and is different from one MO class to another.

The management system controller initializes the AM process on the node (AMnodeIP). The AM first executes the AM_initialization algorithm with the information passed to it from the management controller: address of the policy repository, peer list, parent AM, child list, and new AM initial profile policy. Algorithm 5 depicts the initialization code for the new AM when it first runs.

Algorithm 5. AM Initialization.

AM_Initialization:
Input parameters: ProfilePolicyId, AM_Parent, AM_Peer_List , AM_ChildList, configVector, PolicyRepositoryId

```
1.  begin
2.      call AM_SetPolicySet
3.       AMID=ManagementController.generatingIDforNewAM()
4.       initialize MOProperties with (configVector)
5.       initialize SLA Sensor parameter with (configVector)
6.      //Sending initialize message to AM_Parent.
7.       SendMsg (Initialize, AMID, IAMCHILD )
8.      //Sending initialize message to AM_ChildList.
9.       SendMsg (Initialize, AMID, IAMPARENT )
10.     //Sending initialize message to AM_Peer_List.
11.      SendMsg (Initialize, AMID, IMPEER )
12.     // creating threads for planning and management of the A
13.     call Mng_Loop
1.  end
```

Algorithm 6. Setting AM Policy Set

AM_SetPolicySet:
Input parameters: ProfilePolicyId

```
1.  begin
2.      AM_PolicySet= QuesryPolicyRepository(ProfilePolicyId)
3.  end
```

The initialization module first queries policy repository (Algorihtm 6. *AM_SetPolicySet*) to get the policy set of the initial policy profile (i.e. given as ProfilePolicyId). The AM initialization module invokes a service in management controller to get a unique ID (AMID) for identifying the AM in the management system; the management controller updates Table 6-4, e.g. for future use, such as for AM termination. This identifier is then use to inform the parent and peers about the new AM.

The AM initialization module sends messages to AMs on the parent, children and peer lists to inform them about the new AM in the management system. Note that the AM sends an initialization message to its parent, children, and peer as specified in its input parameters. After this, the AM is ready for its management task. Management loop (*Mng_Loop)* will be last step in AM initialization.

6.5. AM Management Loop

After AM initialization, all the environment variables are initialized and the management loop starts to run. The AM management loop checks for incoming messages and events then takes actions. An AM can take different actions upon the occurrence of an event according to its *active policy profile*. The management loop is the thread inside the AM which runs during the life cycle of the AM. Note that the management loop updates its sensors at the beginning to make sure the MO is alive. Note that in it just asks for one of the sensors value not all of them. If for any reason the sensor update failed, the AM tries for a number of times and after that the AM considers its MO to be no longer alive and it has finished its job. The AM will then terminate itself. In the *Mng_Loop* algorithm, the number of failed tries is compared with maxTry values, this value is one of configuration parameters of the management system that should be specified by the administrator. Details on AM termination is discussed in Section 6.8.

Algorithm 7. AM Management Loop

```
Mng_Loop
Input parameters:  AM_Parent, AM_Peer_List, AM_ChildList
1.  while(true)
2.  begin
3.   update status = Update sensor values
4.   if update status is failed and FailedTry >  maxTry
5.       invoke AM Termination Algorithm
6.   end
7.   if update status is failed and FailedTry <  maxTry
8.       increase FailedTry
9.   end
10.  while (!messageQue.isEmpty())
11.   begin
12.       msg=messageQue.dequeue
13.       Call Handle_message
14.   end
15. //all triggered event are put in a queue
16.   while(!eventQu.isEmpty() )
17.   begin
18.       EV= eventQu.dequeue
19.      for (all PL ∈ PolicySet)
20.       begin
21.           If(PL.Event=EV)
22.             Update sensor PL.condition
23.             If(PL.Condition = True)
24.                    for( all A ∈ PL.ActionList)
25.                  begin
26.                     Execute A
27.                  end
28.             end
29.           end
30.       end
31.   end
32.  end
33. end
```

The list of peer, children, and parents are given to the management loop algorithm. The algorithm first checks its message queue and fetches messages and then does something based on message type. The *Handle_message* algorithm presents the details of the

message handling. Before evaluating a policy, the specified sensor in the policy needs to be updated, which is done in the management loop.

For each type of message (refer to Table 5-2) there is a corresponding action. If the message is an initialization message, then depending on its privilege value the new node will be added to the appropriate list. If the message is a destroy message, the AM will remove the AM identified in the destroy message from all of its internal lists. If the message is a request for updating a heartbeat, then a reply message will be sent back to the requesting AM with updated values of the heartbeat variables. If the message is an update heartbeat message, then depending on its *eventFlag* part of the message (for details refer to 5.4) the receiver AM will send back the heartbeat values or update its event queue. If the message is a "ChangeProfilePolicy", then the policy repository is queried to obtain the new set of policies.

Algorithm 8. Handling Messages

```
Handle_message:
Input parameters: message
1.  begin
2.    Switch(message.Type)
3.    Case Initialization
4.          If message. PrivilegedCode =IAMCHILD
5.               ChildList.Add(message.getSender())
6.          If message. PrivilegedCode =IAMPEER
7.               PeerList.Add(message.getSender())
8.          If message. PrivilegedCode =IAMPARENT
9.               Parent=message.getSender()
10.   Case Destroy
11.         RemoveFromAllList(message.getSender())
12.  Case ReqForHeartbeat
13.         //update heartbeat value
14.         SendMsg (UpdateHeartbeat,message.getSender(),heartbeatValue)
15.  Case UpdateHeartbeat
16.       If message.getEventFlag()= EventCode
17.          //means that sender wants to inform the AM
18.               Update the event queue.
19.       If message.getEventFlag() != EventCode
20.         Update the heartbeat value for the specified sensor
21.   Case ChangeProfilePolicy
22.         profilePolicyId=message.getProfileId()
```

23. Call AM_SetPolicySet algorithm
24. End

6.6. Event Handling

Event handling is done in the management loop. Inside each AM there is an event queue. There are two kinds of events in an AM: internal events and external events. Internal events are timer triggers. External events are those that happen in other managers and the manager is notified by messages will help the manager to notify each other about the events.

How does a manager will trigger an event in another manager? The AM in need or the AM that wants to trigger an event sends an UpdateHeartbeat message with eventFlag equals to **EventCode** (code to show that the intention of this message is event trigger not just updating heartbeat based on request) to other AM/s. When an AM receives the event notification message, it will put the received event in the event queue and upon running the management loop the event will be de-queued and will be processed. The management loop goes through each event in the queue and checks its policy set and invokes policies associated with each event.

The next question is when should a manager trigger an event in another manager? The answer is that the notification of an event should be specified in a policy as an action. For example consider an interactive application in which there is an SLA violation. One action may be to inform its system level AM about its SLA violation, the assumption being that the system level AM should have a related policy to handle the application's SLA violation. The following policy shows this logic (interactive application policy).

On Event: Timer Trigger
If (reponseTime >ERT**)**
  Send (UpdateHeartbeat, responseTime, SystemAM, EventCode)
end

As explained in our model for interactive applications, the SLA sensor is responseTime if it is greater than an expected response time (ERT) for specific percent of jobs during last epoch time there is SLA violation. The action in the policy is to send UpdateHeartbeat

message with responseTime value and putting EventCode in the message, specifying the intention of the message is event trigger not just updating its heart beat values.

6.7. Changes in Management System Configuration

This section illustrates how the management system handles any on-the-go changes in the configuration of the management system. For instance, the administrator may decide to not have manager for a class of MOs anymore in the system, or needs to define a new AM for the MO class that did not have a manager. In this case, all objects of this type of MO class are running without manager and now the management system needs to recognize them and then assign a manager to them.

The administrator can change the AM configuration in terms of changing the information of AM table and Topology table. These changes then will be translated to the following changes in the topology of the AMs:

1. Instantiating new AMs for MOs previously configured and were started before, but with no AM created. In doing so, the management system will extract all MOs addresses that exist in the system obtained from *MO Table* (refer to Table 6-2) and create a manager for each of them, with the same procedure as we had for the arrival of a new MO, the only difference is that we already have MOPhysicalInfo. (refer to Table 6-2)

2. Terminating the AMs of previously configured MOs. In this case, the management system needs to get the AMs associated with the MO class by referring to the *AM Table* (refer to Table 6-1). It can then terminate all the AMs of that AM class. This information can be extracted by referring to the *Instance Table* (refer to Table 6-4). Terminating one specific AM will be done by invoking the termination algorithm remotely on that AM (refer to Section 6.8).

6.8. AM Termination

The management system must be able to terminate an AM. Algorithm 8 illustrates the algorithm for the termination of an AM; the input parameter is the identifier for the AM as explained earlier these identifier are stored in Table 6-5. The termination algorithm could be invoked directly by administrator (it means that the associated MO no longer needs to be managed e.g. for performance reasons) or by the AM itself when its corresponding MO finishes its job. In either case the id of the AM will be extracted from Table 6-5.

In most cases, however, the AM will terminate when the MO that it is managing disappears. How is the AM informed that its MO is done? The answer is that in the management loop, the first thing that AM does is checking its MO sensors(it can be one sensor of the MO sensors list), if nothing is returned from the MO, then after a number of tries, the AM can assume that the MO is gone and can terminate itself.

Termination of an AM means informing all of its neighboring AMs in the management system that the AM is terminating. If the terminated AM is a parent of other managers in the system, then after the AM termination they will continue working based on their policy and nothing will change with respect to their other neighbors. The management system may have its own strategy to somehow reconnect the children to the rest of the management system topology, e.g. to other "parents"; this is a topic for future work.

Algorithm 9. AM Termination

AM_Termination:
Input parameters: AMID

1. begin
   For all node in AM_Parent, AM_ChildList, AM_Peer_List
2. SendMsg(Destroy,AMID)
3. end

6.9. Administrator and management system

After the management system start-up, the next aspect is considering the intervention of the administrator on the behavior of the management system. The administrator should be able to change the behavior of management system. The following describes a number of activities that the administrator may undertake.

- Changing the policy profile of any AM or set of AMs in the management system.
- Terminating an AM in the management system topology.
- Defining a new set of policies/changing the current available policies and updating the policy repository.

One can imagine that the interface between the management system and the administrator can be some kind of "management" panel or "configuration" panel.

The following table provides some suggested configuration commands from the administrator to change the management system.

Table 6-6. Administrator commands

Administrator Command	Description	Back end instructions to run the command
ChangeProfPolc(AMID,ProfPlcID)	Change the policy profile of AM with AMID to ProfPlcID.	- Update profile policy ID of the AM with AMID id. - Connect to the policy repository - Request policy repository to get the new set of policies - The AM downloads the new set of policies.
Terminate AMID	Terminate the AM and return its associated compute nodes to the data center resource pool	- invoke termination module on AMID
addNewPlcy newPolicy profilePolicyID	Administrator defines a new policy to be inserted to profilePolicyID profile policy	- connect to the policy repository - insert new policy to the data base

6.10. Summary

Implementation and operation of the management system had covered in this Chapter. The administrator defines MO classes, AM classes and the topology of the management system. The administrator runs the start-up management system code on the management system root node; then the root initially reads configuration file and initiates managers for each class of management system. Upon arrival of a new MO in the data center the management system instantiates a manager for it. The management system root maintains the list of current MO from all defined class in the data center. In the next Chapter we evaluate the management system in our simulated environment.

Chapter 7

Management System Evaluation

To evaluate the capabilities of our proposed management system and to illustrate its potential impact, in this Chapter we consider the use of the management system in a hypothetical data center. We consider a number of scenarios with different management configurations, scenarios with and without management, and compare them.

We need to simulate the data center behavior and then evaluate the impact of the management model on it. To do this we make use of a data center simulator. We first described our data center simulator and then explain the different management scenarios in terms of the logical configuration of systems, applications, sets of managed objects (MOs), autonomic manager (AM) classes, the topology of AMs and then describe the policies.

Our goal in the designing the scenarios are:

- Demonstrating the simulator and the impact of using simple policies to achieve management goals (Scenario #1).
- Demonstrating the impact of multiple AMs in a hierarchical arrangement (Scenario #2).
- Demonstrating the impact of multiple AMs operating in a combined hierarchical and peer to peer arrangement and where the management system adapts dynamically (Scenarios #3).

The scenarios are compared in terms of energy consumption and SLA violations. Computing data center energy consumption depends on its thermal model and is data center layout. In this research we have used one data center physical layout and its corresponding thermal model for our simulated scenarios. Note that any changes in the data center layout, directly impact the thermal model.

7.1. Data Center Physical Layout

The thermal model used in this research is the Arizona State University thermal model[7] , which is a well-known thermal model in the literature. The data center has two rows of industry standard 42U racks arranged in a typical cold aisle and hot aisle layout. The cold air is supplied by one computer room air conditioner. There are ten racks and each rack is equipped with five 7U (12.25-inch) chassis, totally there are 50 chassis; each of them has a server inside.

The number and specification of servers in each chassis in terms of number of cores and its CPU parameters is configurable. The physical layout of the ASU HPC datacenter is shown in Figure 7-1. The ASU datacenter's physical dimensions are $9.6m \times 8.4m \times 3.6m$.

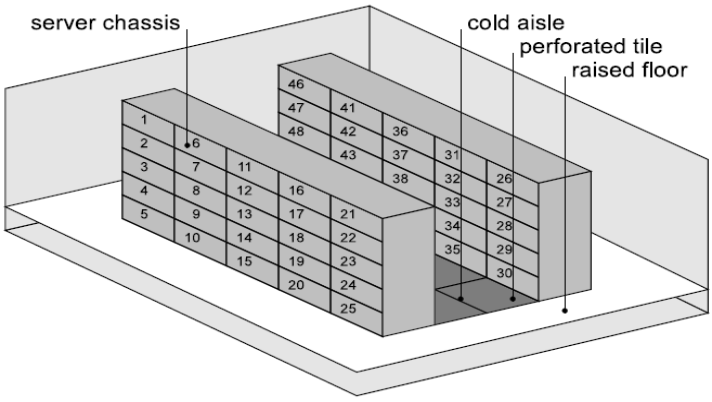

Figure 7-1. Physical layout of ASU datacenter.

The cooler used in our model is HP research cooler, which is high cited cooler in the literature. Before getting to specification of the cooler, we need to look at how cooler performance is calculated.

The efficiency of a Computer Room Air Conditioner (CRAC) depends on air flow velocity and conductivity of materials which is quantified as the Coefficient of Performance (COP). In other words, the COP measures how much heat is removed by consuming a unit amount of heat (heat as energy). Therefore, the higher the COP value

means more efficient the cooling system [15]. The COP for a cooling system is not constant and will increase with increasing the air temperature. Figure 7-2 shows how the COP changes over temperature as published by Hewlett-Packard (HP) laboratories [15]. This is a well-known graph in the simulation of energy consumption.

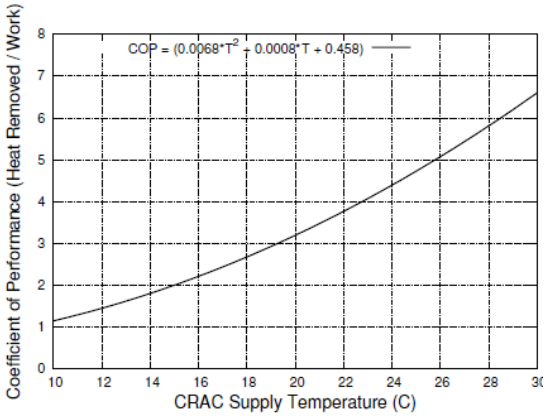

Figure 7-2. COP-Temperature of HP Lab CRAC Units.

For COP changes over temperature, we have used the HP water-chilled CRAC unit change rate which is:

$$COP = 0.0068*T^2+0.0008*T+0.458.$$

where:

T is the outlet temperature of the cooler.

Based on this formula, it is better to maintain the temperature of the air coming into the CRAC as high as possible. *But how high?* For an upper limit on the CRAC output temperature, the nominal temperature at which devices work reliably is a suitable bound (also called the redline temperature). The redline nominal value is between 25C-35C.

7.2 Data Center Energy Consumption

The next issue to address is how to compute the power consumption of the data center having its Thermal Model and the COP.

In the CFD modeling of a data center [1, 3], heat distribution between computing nodes is modeled as a Heat Recirculation Matrix (HRM) $D= \{d_{ij}\}_{N\times N}$. Each entry in the matrix shows the coefficient of distributed heat from node i to node j. Figure 7-3 shows an HRM with the coefficient of re-circulated heat for each i and j server. Note that large values are observed along diagonal and that there is strong recirculation among neighboring servers.

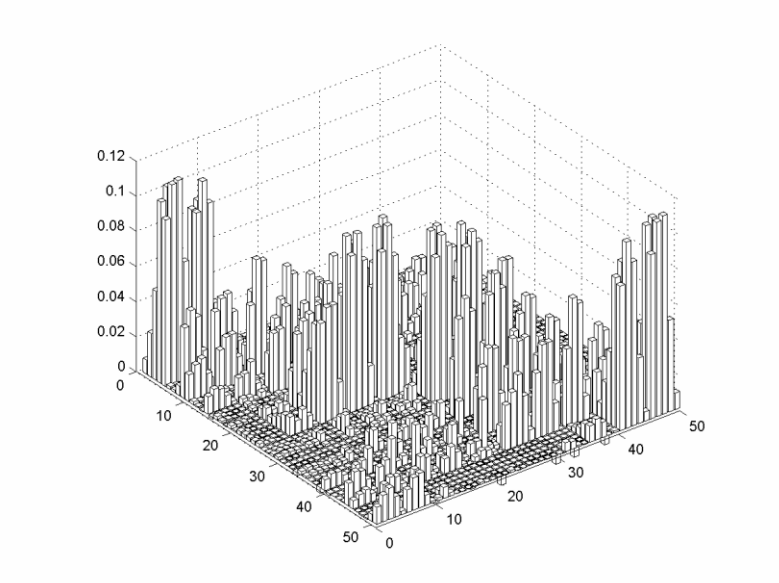

Figure 7-3. Rrecirculation Factor Between Servers.

Next figure shows how the temperature of a node can be calculated:

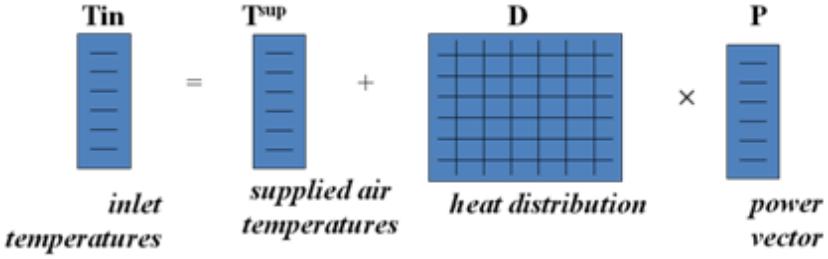

Figure 7-4. Node temperature calculation.

For the total system:

$$P^{Computing} = \sum_{i \text{ all nodes}} P_i \qquad (3)$$

$$P^{Cooling} = \frac{p^{Computing}}{COP} \qquad (4)$$

where:

P_i : Power consumption of node i

$P^{Computing}$: Total computing power of all nodes

$P^{Cooling}$: Total cooling power of all nodes

For calculating the COP, the maximum temperature of computing nodes and red line temperature are needed, here they are computed as follows [7]:

$\{T_i^{in}\} < T^{red}$ for all chassis's

Or $\max\{T^{in}\} < T^{red} \rightarrow T^{sup} + \max\{DP_i\} < T^{red}$

1. $T^{sup} < T^{red} - \max\{DP_i\}$

$$p^{Cooling} = \frac{p^{computing}}{COP(T^{red} - \max\{DP_i\})}$$

$$p^{Total} = p^{computing}\left(1 + \frac{1}{COP(T^{red} - \max\{DP_i\})}\right)$$

where:

T^{red}: Maximum tolerable temperature of nodes.

T_i^{in}: Inlet temperature of node i.

T^{sup}: Cooler supply temperature.

D: heat recirculation matrix.

P_i : Power consumption of node i.

Since CPU is the main power consuming component of a compute node, our simulator is designed in such a way that it calculates the CPU utilization of all compute nodes in the data center. The simulator calculates cooling and computing power of the whole data center given the CPU utilization of all the compute nodes and thermal mode.

For each server, the computing power is proportional to CPU utilization. To compute the energy consumption of the data center, we just consider the summation of power consumption during the simulation time.

Figure 7-5 shows the black box model of the thermal model which the simulator is designed to deal with. Thermal specifications of coolers, the physical specification of servers, and the data center are inputs.

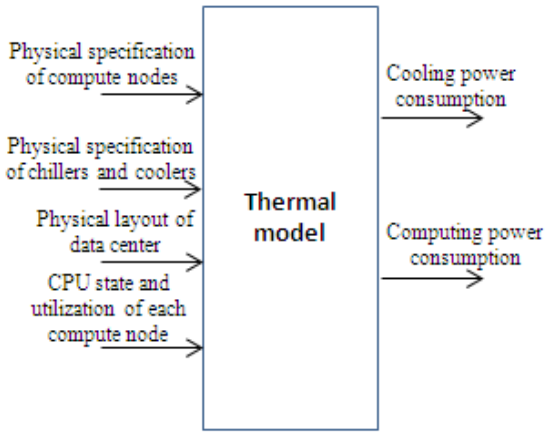

Figure 7-5. Thermal model.

7.3 Scenario #1: Single Manager and Scalable Data Center

In this scenario, our simulated data center has the same layout as explained in previous section, except each chasses has five compute node (in total our data center has 250 compute nodes); simulated servers are Proliant HP DL320. This server has a standby power consumption of 5Watts; when its idle power consumption is 100 Watts and with a fully utilized the CPU consumes 300 Watts. The DL320 has an Intel® Xeon® E3-1200v2 processor which has frequency scaling levels which are 3.07, 3.2, and 4.2 GHz, which when normalized to the "base level" are 1, 1.07 and 1.37 (refer to the definition of compute nodes in Definition 12.)

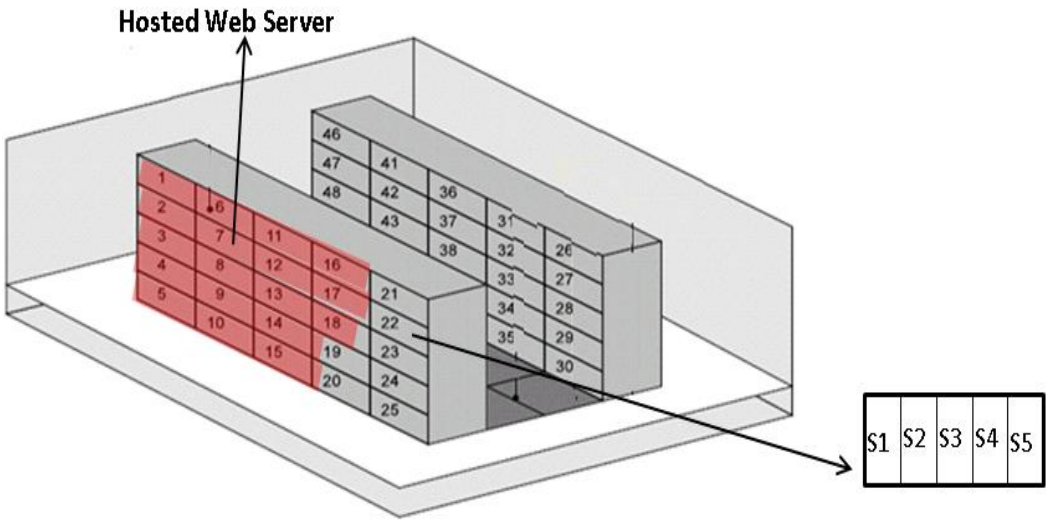

Figure 7-6. Data Center Layout in Scenario # 1 (Each chassis has five servers).

7.3.1 Managed Objects and Classes in Scenario #1

Table 7-1 shows the configuration of the physical layout of the data center. This layout will also be used for all scenarios in this Chapter.

Table 7-1. Physical Data Center Set of Manageable Objects

dataCenterT	**SysList**={entSys} **Racks**={RackT} **RackT**={r1,r2,r3,r4,r5,r6,r7,r8,r9,10}

	r1 = {ch11,ch12,ch13,ch14,ch15} r2 = {ch21,ch22,ch23,ch24,ch25} r3 = {ch31,ch32,ch33,ch34,ch35} r4 = {ch41,ch42,ch43,ch44,ch45} r5 = {ch51,ch52,ch53,ch54,ch55} r6 = {ch61,ch62,ch63,ch64,ch65} r7 = {ch71,ch72,ch73,ch74,ch75} r8 = {ch81,ch82,ch83,ch84,ch85} r9 = {ch91,ch92,ch93,ch94,ch95} r10 = {ch101,ch102,ch103,ch104,ch105} ch11= {n111,n112,n113,n114,n115} ch12= {n121,n122,n123,n124,n125} ch13= {n131,n132,n133,n134,n135} ch14= {n141,n142,n143,n144,n145} ch15= {n151,n152,n153,n154,n155} ch21= {n211,n212,n213,n214,n215} ch22= {n221,n222,n223,n224,n225} ch23= {n231,n232,n233,n234,n235} ch24= {n241,n242,n243,n244,n245} ch25= {n251,n252,n253,n254,n255} ch31= {n311,n312,n313,n314,n315} ch32= {n321,n322,n323,n324,n325} ch33= {n331,n332,n333,n334,n335} ch34= {n341,n342,n343,n344,n345} ch35= {n351,n352,n353,n354,n355} ch41= {n361,n362,n363,n364,n365} ch42= {n421,n422,n423,n424,n425} ch43= {n431,n432,n433,n434,n435} ch45= {n441,n442,n443,n444,n445} ch45= {n451,n452,n453,n454,n455} ch51= {n511,n512,n513,n514,n515} ch52= {n521,n522,n523,n524,n525} ch53= {n531,n532,n533,n534,n535} ch54= {n541,n542,n543,n544,n545} ch55= {n551,n552,n553,n554,n555} ch61= {n611,n612,n613,n614,n615} ch62= {n621,n622,n623,n624,n625}} ch63= {n631,n632,n633,n634,n635} ch64= {n641,n642,n643,n644,n645} ch65= {n651,n652,n653,n654,n655} ch71= {n711,n712,n713,n714,n715} ch72= {n721,n722,n723,n724,n725} ch73= {n731,n732,n733,n734,n735} ch74= {n741,n742,n743,n744,n745}

	ch75= {n751,n752,n753,n754,n755} ch81= {n811,n812,n813,n814,n815} ch82= {n821,n822,n823,n824,n825} ch83= {n831,n832,n833,n834,n835} ch84= {n841,n842,n843,n844,n845} ch85= {n851,n852,n853,n854,n855} ch91= {n911,n912,n913,n914,n915} ch92= {n921,n922,n923,n924,n925} ch93= {n931,n932,n933,n934,n935} ch94= {n941,n942,n943,n944,n945} ch95= {n951,n952,n953,n954,n955} ch101= {n1011,n1012,n1013,n1014,n1015} ch102= {n1021,n1022,n1023,n1024,n1025} ch103= {n1031,n1032,n1033,n1034,n1035} ch104= {n1041,n1042,n1043,n1044,n1045} ch105= {n1051,n1052,n1053,n1054,n1055} where $n_{i,}$ is a nodeT **ThermalModel** =< coolerList, RedTemp, ThermalMap > = <{coolerT}, 35, DMatrix>
nodeT	**MIPS**={1 1.04 1.4 } **FullLoadPowerConsumption**= {300, 336, 448} **ZeroLoadPowerConsumption**={100, 100, 128} **StandbyPower**= 5
coolerT	**CoP_Equ=** $0.0068*T^2+0.0008*T+0.458$

In this experiment, a web server is hosted in the data center and 90 compute nodes are allocated to the web server. The workload for this web server is a scaled up workload based on the web server workload for the world cup 1999 traffic [20]. Note that original workload log has 9500 http request arrivals per second; in our experiment we scaled up the workload to have, an average 10 times of the original workload i.e. 95000 http requests per second. The workload is specified via file of jobs (see Table 7-2, a sample of the workload (Workload.txt) is provided in Appendix B).

We consider the system to be one enterprise system, which contains one enterprise application – a web server. A manager is attached to the web server which monitors its SLA and takes actions (see Table 7-8) based on its active policies. For this web server, we consider an SLA violation to occur when the response time is more than two seconds for 90% of workload that has been done in last epoch time (here 60 seconds). Two different profile policies (SLA and Green) have been defined and are compared in terms

of the total number of SLA violations and total energy consumption. The following tables contain configuration parameters to set up the management system.

On top of the physical objects, we have modeled an Enterprise system and Enterprise application. Table 7-2 shows the definition of this system and application based on our definitions in previous chapters. The enterprise system has a single resource manager with MHR is its resource allocation algorithm (Minimum Heat Recirculation; see Algorithm 2.) which utilizes all nodes of the system. First come first served (FCFS) is the scheduling algorithm in the system. The application makes use of all 90 nodes.

Table 7-2. Manageable Objects Scenario#1.

Manageable Object	
entSys	**Id=1** **Type**=Enterprise **Workload**= {entApp} **Queue** **ResManagers**= {MHR) } **Sched**= {FCFS} **Racks**= {r1,r2,r3,r4}
entApp	**Id=1** **startTime**=1 **duration**=0 **computeNodes**:<90,90> **computeIntensity**=(1,1000) **workload**=WorkLoad1.txt **SLA**=(2,90%,60s) terminationTime=0

In this scenario we have EntAppClass and NodeClass (refer to Table 7-3) as classes of managed objects in this scenario. NodeClass is defined since the actuators of application need access to change the status of nodes belonging to the application. Therefore, we define a class for nodes which just has actuators and there is no manager attached to it. The managed object associated with EntAppClass and NodeClass are application entApp and nodeT. EntAppClass has sensors, events, and actuators. NodeClass just has a set of actuators for changing the CPU working frequency and status (sleep or active).

Application level MO class EntAppClass actuators are as follows:

- increaseFreqFullyUtilized(id): which increases the frequency of all fully utilized nodes of application with given id.
- activateAllSleepNodes(id): which activates all sleeping nodes of application with given id.
- activateHalfSleepNodes(id): which activates half of the sleeping nodes of application with given id.
- increaseFreqBusyNodes(id): which increases the frequency of all busy nodes of application with given id.
- decreaseFreqAllBusyNodes(id),in which decrease frequency of all busy nodes of application with given id.
- sleepAllidleNodes(id): which puts to sleep all idle nodes of application with given id.

EntAppClass has an internal timer (Timer1) when triggered causes the managed object class event. Note that this experiment shows one of the implications of MOProperties; timer value is one of MOProperties variable.

According to our model, for an enterprise application with given id, the response time for the last epoch time window is defined as an SLA sensor i.e. reponseTimeInLastEpoch(id) in the definition of EntAppClass.

Table 7-3. Managed Object Classes (EntAppClass, NodeClass) for Scenario#1

EntAppClass	**MOs= {**entApp} **Event**= {Timer1TriggerEvent} **Sensor**={ reponseTimeInLastEpoch(id)} **Actuator**={ increaseFreqFullyUtilized, activateAllSleepNodes, activateHlafSleepNodes, increaseFreqBusyNodes, decreaseFreqAllBusyNodes, sleepAllidleNodes} **MOProperties**={ (Timer1Vlaue,60s)}
NodeClass	**MOs= {**nodeT} **Event**= { } **Sensor**={ } **Actuator**={ increaseFreq(id), activateNode(id),SleepNode(id),decreaseFreq(id)} **MOProperties**={ }

Table 7-3 shows managed object class in the simulation scenario i.e. EntAppClass and NodeClass. The list of all instances of AM classes which, in this case, are associated with managed object classes is illustrated in Table 7-6. In this scenario, we just have one AM i.e. AMEntAppClass and no pair. The manager, in this scenario has two policy sets.

Following tables show data center set of manageable objects and classes and their associated AM classes, and the topology of the management system in this scenario.

Table 7-4. Data Center Set Of Manageable Object.

Set of Manageable object
MO_{DC}={dataCenterT}
MO_{Racks}={ rackT}
MO_{node}= {nodeT}
$MO_{EntrpriseSys}$= { EntSys}
MO_{Apps}= {entApp}

Table 7-5. Data Center Managed Object Class.

Managed Object Class
$MOClasses_{DC}$= { EntAppClass, NodeClass }

Table 7-6. AM Classes and MO Classes.

AM Class	**MO class**
AMEntAppClass	EntAppClass, NodeClass

Table 7-7. AM Pairs and Management Topology.

AMPair	{}
T_{DC}	V={ AMEntAppClass } E=AMPair

Our managed data center is then defined as:

$$MDC < dataCenterT , T_{DC} >$$

with

$$AMClasses_{DC} \text{ and } MOClasses_{DC}.$$

7.3.2 Scenario #1 Policies

Table 7-8 defines the policies for manager $AM_{EntAppClass}$ attached to the application EntApp. EntAppClass1 policies are time-triggered policies. Timer1TriggerEvent is the event of policy. Timer1 gets initialized in time of initialization of AM for the MO class via the management system controller and as shown in Table 7-3 the timer is set to 60 seconds.

There are two sets of policies defined for $AM_{EntAppClass,}$ (Green and SLA). The Green policy profile aims to minimize energy consumption while the SLA policy profile tries to minimize SLA violations. In the SLA policy profile, the autonomic manager $AM_{EntAppClass}$ checks for any SLA violations and tries to increase CPU clock frequency and also activate all sleeping nodes. If the frequency scaling is not supported by the compute nodes this part (frequency scaling → activating nodes) fails and the second part of the action (which comes after "|" operator) of the policy is executed, namely, activating sleeping nodes (refer to Table 5-1 for operator details) .

The Green policy profile also does dynamic frequency scaling and activation/deactivation of compute nodes if SLA violations happen. This policy simply checks to see if there is an SLA violation and then increases the clock frequency of the fully utilized nodes and activates half of sleeping nodes assigned to the application. If there is no SLA violation it decreases clock frequency of all nodes that running jobs.

SLAViolationCheck is a function that checks if 90% of reponseTimeInLastEpoch vector is less than 2 seconds and returns 1 otherwise 0. Epoch time is one of enterprise application defined parameters (refer to Definition 6. and Table 7-2).

Note that reponseTimeInLastEpoch is defined in the list of EntAppClass MO class sensors. Last argument of SLAViolationCheck function is application id, in our scenario we just have one application with id=1.

Table 7-8. Policy definition for Scenario#1

SLA Profile For AM Attached to Enterprise App	PL0:
	On Event: **Timer1TriggerEvent**
	If (SLAViolationCheck (90%,reponseTimeInLastEpoch,2, id)>0)
	begin
	increaseFreqBusyNodes(id) → activateAllSleepNodes(id) | activateAllSleepNodes(id)
	End

Green Profile For AM Attached to Enterprise App	PL1:
	On Event: **Timer1trigger**
	If (SLAViolationCheck(90%,reponseTimeInLastEpoch,2, id)>0)
	begin
	increaseFreqFullyUtilized(id) →activateHalfSleepNodes |activateHalfSleepNodes(id)
	end
	PL2:
	On Event: **Timer1trigger**
	If (SLAViolationCheck(90%,reponseTimeInLastEpoch,2, id)<0)
	begin
	decreaseFreqAllBusyNodes(id) → sleepAllidleNodes(id)| sleepAllidleNodes(id)
	End

The simulation results are shown in Table 7-9. Even with these trivial policies, there is improvement in computing; the application with Green policies consumes less power than the SLA based policies. With the SLA based polices, the total energy consumption (cooling and computing power) is not available through the simulation since the inlet temperature exceeded the red temperature 475 times and as a result the simulator could not calculate the power consumed. In a real data center, the temperature should not exceed the red temperature; in the simulator higher than red temperature causes the cooling power to be negative value.

Table 7-9. Results for Scenario #1

Profile Policy	Green	SLA
Computing power of Web server (Watt)	7.7 * 10^8	9.6*10^8
total energy consumption (Watt × Simulation Time)	1.9*10^9	N/A
Mean power consumption (Watt)	26982	N/A
Number of times crossing red temperature	0	475
SLA Violation	132	38

The main objective of this experiment was to introduce in some detail how the abstract model introduced previously can be used to model a data center, applications, management objects and classes and to illustrate the scalability of data center simulator; it also illustrates the impact of policy based management to manage energy consumption and SLA violations.

7.4 Scenario #2: Hierarchical Autonomic Manager Arrangement

In this scenario, we develop and illustrate a hierarchical arrangement of managers. A hierarchy of managers corresponds well with the expected managed elements in the data center. By having multiple levels of AMs, we aim to get better performance in terms of total energy consumption of the whole data center while trying to minimize the violations to service level agreements (SLA). Prior to getting to discussing details of the second scenario simulation, we illustrate hierarchical data center management.

7.4.1 Hierarchical Management System

A hierarchical management structure can be based on the physical arrangement of elements, from blade server, to rack, to cluster. For instance, a manager (AM) at the cluster level can manage the layer below it, namely the racks. In each rack, a particular AM could then make decisions about, for example, the number of active servers. The

hierarchical management system could have AMs at the cluster level, the server level, and the device level manager; at each level, the AM would have its own set of policies.

The hierarchical structure of managers is not confined to the physical environment; it can be based on the arrangement of applications running in the datacenter, regardless of the physical location of their related compute nodes. For instance, one type of application may be hosted on different racks in the data center, and one AM could be in charge of controlling all compute nodes used by that application. Managers could be assigned to applications with similar workload or even applications with similar user agreements. Given that the AM arrangement is hierarchical, the placement and number of levels in the hierarchy is administrator dependent. Figure 7-7 illustrates a possible hierarchical arrangement of AMs.

We have modeled our prototype management system (illustrated in Figure 7-7), using a three level hierarchy. At the bottom level, we have *local AMs.* Each *local AM* is attached to a number of compute nodes. The second level of AMs are called aggregate *AMs*; they logically aggregate management responsibility from the local level to the data center level. AMs at this level have the AMs at the first level as their managed elements. At the top level, there is one (or more for replication) *Data Center AM* which acts as the coordinator among all *aggregate AMs.*

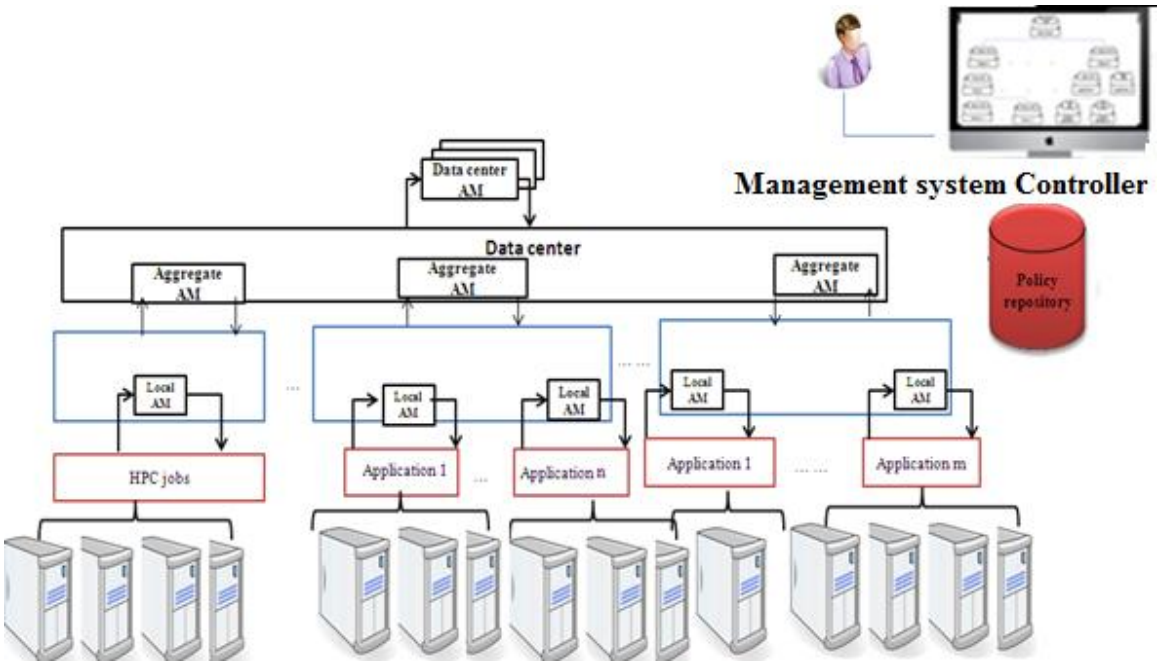

Figure 7-7. Prototype Hierarchical Management System

AMs in our management system exchange information with the other AMs in the levels above or below or with peers (same level AMs). Information is exchanged via messages (refer to Table 5-2), this information is called *heartbeat* and *configVector* data that is the sensor and actuator information. Aggregate level or data center level AMs may inquire of their children for heartbeat updates to make better decisions. Any changes in configuration parameters or policies of the child are then sent from the parent AM as configuration parameters.

7.4.2 Hierarchical HPC System

In this experiment, we have hierarchical arrangement of managers for an HPC type workload. The layout of the data center in this experiment is the same as in scenario one except that in each chassis we assume that we have one HP server and so there are 50 physical servers in total in the system. In terms of systems, the data center in this scenario has two HPC type systems. Table 7-10 illustrates data center set of manageable objects and their configuration parameters in this scenario (the list of systems will be defined in the following). Note that the physical layout of this scenario is used in other scenarios in

this Chapter, so we have the same data center manageable objects for the remaining scenarios except that the system list may change.

Table 7-10. Data Center Set of Manageable Objects

dataCenterT	**SysList**={ hpcSys1, hpcSys2} **Racks**={RackT} **RackT**={r1,r2,r3,r4,r5,r6,r7,r8,r9,10} r1 = {ch11,ch12,ch13,ch14,ch15} r2 = {ch21,ch22,ch23,ch24,ch25} r3 = {ch31,ch32,ch33,ch34,ch35} r4 = {ch41,ch42,ch43,ch44,ch45} r5 = {ch51,ch52,ch53,ch54,ch55} r6 = {ch61,ch62,ch63,ch64,ch65} r7 = {ch71,ch72,ch73,ch74,ch75} r8 = {ch81,ch82,ch83,ch84,ch85} r9 = {ch91,ch92,ch93,ch94,ch95} r10 = {ch101,ch102,ch103,ch104,ch105} ch11= {n11} ch12= {n12} ch13= {n13 } ch14= {n14} ch15= {n15 } ch21= {n21 } ch22= {n22 } ch23= {n23 } ch24= {n24 } ch25= {n25 } ch31= {n31 } ch32= {n32 } ch33= {n33 } ch34= {n34} ch35= {n35 } ch41= {n41 } ch42= {n42 } ch43= {n43 } ch45= {n45 } ch51= {n51 } ch52= {n52 } ch53= {n53} ch54= {n54} ch55= {n55} ch61= {n61 } ch62= {n62}

	ch63= {n63} ch64= {n64} ch65= {n65} ch71= {n71} ch72= {n72} ch73= {n73} ch74= {n74} ch75= {n75} ch81= {n81} ch82= {n82} ch83= {n83} ch84= {n84} ch85= {n85} ch91= {n91} ch92= {n92} ch93= {n93} ch94= {n94} ch95= {n95} ch101= {n101} ch102= {n102} ch103= {n103} ch104= {n104} ch105= {n105} where $n_{i,}$ is a nodeT **ThermalModel** =< coolerList, RedTemp, ThermalMap > = <{coolerT}, 35, DMatrix>
nodeT	**Id** **MIPS**={1 1.04 1.4 } **FullLoadPowerConsumption**= {300, 336, 448} **ZeroLoadPowerConsumption**={100, 100, 128} **StandbyPower**= 5
coolerT	**CoP_Equ=** $0.0068*T^2+0.0008*T+0.458$

We assume that these 50 physical servers are divided into two separate HPC systems; one with 30 compute nodes (HPC2) and the other system has 20 compute nodes (HPC1).

Each of our HPC systems runs an HPC workload[9] consisting of long and short batch jobs (refer to Table 7-11 for more detail about jobs). Each job in the workload has an arrival time, duration, needed CPU utilization (will be used for the thermal model to calculate the power) and deadline (maximum waiting time in the system). The workload file is presented in Appendix B.

Table 7-11. HPC Jobs Specification.

	HPC1	***HPC2***
Number of jobs	200	730
Average duration	500	1098
Average number of compute nodes	2	4
Average CPU utilization	44%	56%

Note that each system has a queue in which jobs are queued upon arrival. Jobs then get dispatched to compute nodes and are executed. The time that a job stays in the queue is the waiting time. An SLA violation occurs when a deadline is passed for a job in the workload.

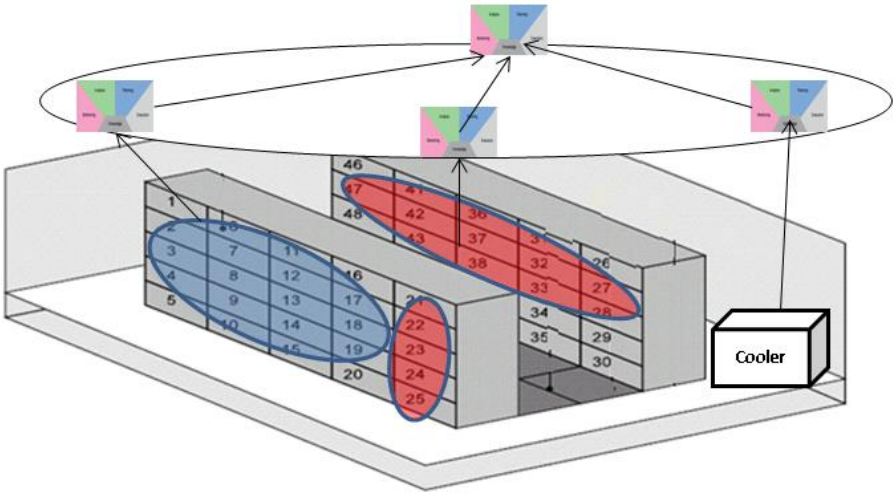

Figure 7-8. Scenario#2: System and Managers Arrangement

9 This log has been used for evaluating resource allocation and scheduling algorithms by the Los Alamos laboratory.[15]

7.5.3 Managed Objects in Scenario #2

Besides the physical objects in this scenario, we have deployed HPC systems as well. Table 7-8 shows the configuration of both HPC systems as manageable objects in the data center. Each of the HPC systems has a single resource manager with the Minimum Heat Recirculation (MRH see Algorithm 2) as its resource allocation algorithm. First come first served (FCFS) is the scheduling algorithm in the systems. The workload data is presented in Appendix B.

Table 7-12. Manageable Objects for HPC Systems in Scenario#2

Manageable Object	
hpcSys1	**Id=1** **Type**=HPC **SysWorkload**= {WorkLoad1.txt} **Queue** **ResManagers**= {MHR} **Sched**= {(FCFS)} **Racks**= {r1,r2,r3,r4}
hpcSys2	**Id=2** **Type**=HPC **SysWorkload**= {WorkLoad2.txt} **Queue** **ResManagers**= {MHR)} **Sched**= {(FCFS)} **Racks**= {r5,r6,r7,r8,r9,r10}

Our set of manageable objects are shown in Table 7-13 and the managed object classes are shown in Table 7-14 .

Table 7-13. Data Center Set Of Manageable Objects.

Manageable Objects
MO_{DC}={dataCenterT}
MO_{Racks}={ rackT}
MO_{node}= {nodeT}
MO_{hpcSys}= { hpcSys1, hpcSys2}
MO_{cooler}={coolerT}

Table 7-14. Data Center Managed Object Classes.

MOClasses$_{DC}$
DataCenterClass
HPCSys1Class
HPCSys2Class
CoolerClass
NodeClass

We have two classes for our HPC systems, HPCSys1Class and HPCSys2Class. For this scenario we have separate classes because we treat one of the systems as a "high priority" system and one as a "low priority" system. This will result in differences in policies and how they are managed. Table 7-15 provides details on the managed object classes HPCSys1Class, HPCSys2Class and DataCenteClass.

Table 7-15. Managed Object Classes: Scenario#2

Managed Object Class	
HPCSys1Class	**MOs**= {hpcSys1} **Event= {** Timer1trigger} **Sensor**={ SLAViolation(id)} **Actuator**={ increaseFreqFullyUtilized(id), activateAllSleepNodes(id), activateHlafSleepNodes(id), increaseFreqBusyNodes(id), decreaseFreqAllBusyNodes(id), sleepAllidleNodes(id) } **MOProperties**={ (Timer1,60s)}
HPCSys2Class	**MOs**= {hpcSys2} **Event= {** Timer1trigger} **Sensor**={ SLAViolation(id)} **Actuator**={ increaseFreqFullyUtilized(id), activateAllSleepNodes(id), activateHlafSleepNodes(id), increaseFreqBusyNodes(id), decreaseFreqAllBusyNodes(id), sleepAllidleNodes(id) } **MOProperties**={ (Timer1,60s)}
DataCenterClass	**MO=** {dataCenterT} **Event= {**SLAViolationSys1, SLAViolationSys2, SysBlockTimer-Trigger, RedTemeratureEvent } **Sensor**={ } **Actuator**={ blockSys(id), unblockSys(id)} **MOProperties**={(SysBlockTimer ,120)}
CoolerClass	**MO=** {coolerT} **Event=** **Sensor**={inletTemperature} **Actuator**={} **MOProperties**={}

NodeClass	**MOs= {**nodeT} **Event**= { } **Sensor**={ } **Actuator**={ increaseFreq(id), activateNode(id),SleepNode(id),decreaseFreq(id)} **MOProperties**={ }

The classes of HPC systems has SLAViolation(id) sensors. This sensor checks every Timer1Trigger if there is an SLA violation for HPC job runs inside the system (with given id) since last Timer1Trigger (recall that an SLA violation for HPC jobs means terminationTime is less than the time the job actually finished). SLAViolation(id) will return one if there was a violation. Otherwise, if there is no SLAViolation for jobs run since last timer1Trigger event then SLAViolation will be zero. To understand better how SLAViolation gets valued, consider what is happening in each management cycle: the manager will just ask for one sensor value to make sure that the MO is still alive. The manager also checks the event queue (in this case time1Trigger will be in the queue) and updates all sensors related to policies that their events has triggered which in this case is SLAViolation(id). So, SLAViolation(id) will get called upon trigger of Timer1.

There are also actuators defined as part of the classes; their descriptions are as follows:

- increaseFreqFullyUtilized(id): which increases the processor frequency of all fully utilized nodes of the system with given id (if possible).
- activateAllSleepNodes(id): which activates all sleeping nodes of the system with given id.
- activateHalfSleepNodes(id): which activates half of the sleeping nodes of the system with given id.
- increaseFreqBusyNodes(id): which increases the frequency of all busy nodes of the system with given id.
- decreaseFreqAllBusyNodes(id),in which decrease frequency of all busy nodes of the system with given id.
- sleepAllidleNodes(id): which puts to sleep all idle nodes of the system with given id.

The HPC system managed object class has a MOProperties parameter: Timer1 with a given value of 60 seconds; this is the value for the timer trigger event.

The data center managed object class has four associated events; three of them are raised in the data center AM through messages from other managers in the system: one from the AM connected to the cooler, two others from managers of the HPC systems; these are specified in the following.

Note that in the Event Handling section, we explained that if a manager wants to inform another manager in the system about its need or its critical situation it sends an UpdateHeartbeat message with an EventCode in it. Usually, the receiver should define an event corresponding to this type of messages. In this scenario, upon an SLA violation in either system their AM sends an EventCode message to the data center AM. Note that SLAviolation is checked every Time1Trigger time. The data center AM will handle that as an event and will check to see if there is a related policy. There is another EventCode message for the data center MO class which comes from the AM attached to the cooler i.e. RedTemeratureEvent. Another event for the data center MO class is its internal timer which is set when a system is blocked; its trigger is an internal event.

The data center class actuators are for blocking the system (blockSys (id)) and unblocking a system (unblockSys(id)); Id is the identifier of the HPC system. "Blocking" a system means that the system should queue arriving jobs and not start running any additional ones; there is no change in the existing workload, i.e., jobs already executing would continue to execute. Blocking/Unblocking a system is done through the system scheduler; this means that the scheduler starts to queue jobs upon receiving blocking request. Note that the communication with scheduler is not the focus of this thesis.

The Data Center class has an SLA sensor for each of its HPC systems which obtain its values from HPC system sensors.

The cooler class sensor provides its inlet temperature (inletTemperature).

The managed object classes are illustrated in Table 7-15. There are two managed object classes for the HPC systems, HPCSys1Class and HPCSys2Class (hpcSys1 and hpcSys2 as their MOs), one DataCenterClass, and CoolerClass.

For the management of this data center, we assume that we have an AM to manage each HPC system and that we have a data center level AM. We also have an AM attached to the cooler that directly communicates with data center level AM.

Table 7-17 shows the AM classes and the classes of objects that they are associated with, the AM pairs and topology.

Table 7-16. AM and MO Classes

AM Class	**MO class**
HPCSys1AM	HPCSys1Class, NodeClass
HPCSys2AM	HPCSys2Class, NodeClass
DataCenteAM	DataCenterClass, HPCSys1Class, HPCSys2Class
CoolerAM	CoolerClass

Table 7-17. AM Pair and Management Topology Scenario#2

AMPair	(DataCenterAM, HPCSys1AM,1), (DataCenterAM, HPCSys2AM,1), (DataCenterAM, CoolerAM,1)
T_{DC}	V={ HPCSys1AM, HPCSys2AM, DataCenteAM, CoolerAM} E=AMPair

For Scenario #2, we consider two policy profiles, Green and SLA, similar to Scenario #1. These respectively try to minimize energy consumption and minimize SLA violations.

Table 7-18. Policy Definitions for Scenario#2

Policy Profile Name	AM	Policy Definition

SLA	HPCSys1AM and HPCSys2AM	PL0:
		On Event: **Timer1trigger**
		If (SLAVilation(id))
		begin

		Send message (UpdateHeartbeat, DataCenterAM, SLAViolation, EventCode)
		End
		
		PL1:
		On Event: **Timer1trigger**
		If (SLAViolation(id))
		begin
		increaseFreqBusyNodes(id) → activateAllSleepNodes(id) | activateAllSleepNodes(id)
		End

Green	HPCSys1AM and HPCSys2AM	PL2:
		On Event: **Timer1trigger**
		If (SLAViolation(id))
		begin
		Send message[10] (UpdateHeartbeat, DataCenterAM, SLAViolation,EventCode)
		End
		
		PL3:
		On Event: **Timer1trigger**
		If (SLAViolation(id))
		begin
		increaseFreqFullyUtilized(id)→activateHalfSleepNodes(id)|activateHalfSleepNodes(id)
		End
		PL4:
		On Event: **Timer1trigger**
		If (SLAViolation(id)==0)
		Begin
		decreaseFreqAllBusyNodes(id)→ sleepAllidleNodes(id)| sleepAllidleNodes(id)
		End

	Cooler Policy	PL5:
		On Event: true
		If (Max temperature is greater than Red temperature)
		begin
		Send message (UpdateHeartbeat, DataCenterAM, ,RedTepmeratureEvent,EventCode)
		End

	Data Center	PL6: On Event: **SysBlockTimerTrigger**

[10] Each AM as part of the management system has an interface to send and receive message.

		if (true)
		begin
		unblock(1)
		end
		
		PL7:
		On Event: (**SLAViolationSys$_1$ | SLAViolationSys$_2$**)
		if (SLAViolation(1))
		begin
		Send message (ChangePolicyProfile , hpcSysAM1, SLA based)
		end
		else
		begin
		Send message (ChangePolicyProfile , hpcSysAM1, Green)
		end
		
		PL8:
		On Event: (**SLAViolationSys$_1$ | SLAViolationSys$_2$**)
		if (SLAViolation(2))
		begin
		Send message (ChangePolicyProfile , hpcSysAM2, SLA based)
		end
		else
		begin
		Send message (ChangePolicyProfile , hpcSysAM1, Green)
		end
		
		PL9:
		On Event: **RedTemperatureEvent**
		if (true)
		begin
		block(1)
		End

System level *SLA* policy profile has time-triggered policies (every 60 seconds) PL0 and PL1. Upon timer trigger, the AM checks for SLA violations of its associated system and if there is, it tries to do dynamic CPU frequency scaling and activate sleeping compute nodes. If compute nodes do not support the frequency scaling, this policy just activates sleep nodes. PL0 reports SLA violation to the data center level AM as an event. Data center AM has policies using this event (PL7 and PL8).

System level *Green* policy profile has time-triggered policies (every 60 seconds) PL2, PL3, and PL4. Policies tries to do dynamic frequency scaling and activation/deactivation of compute nodes if SLA violations happen. With this profile, the AM tries to keep active compute nodes and CPUs at moderate frequency levels based on whether there are SLA violations or not.

In both policy profiles, on any SLA violation, the manager at the system level will inform the manager at the data center level about the situation by sending the update heart beat message with *EventCode*. The data center AM, upon receiving the *EventCode* message, will queue it and then process policies in data center level AM that depend on this event (refer to PL7, PL8 in Table 7-18).

The data center level policy depends on events, i.e. from the HPC systems (refer to policies PL7 and PL8: SLAViolationSys1 and SLAViolationSys2). According to our model to have status of a MO as a sensor for its AM; corresponding managed object should be in the list of MO of the manager. Therefore, we define hpcSys1 and hpcSys2 as managed objects for data center managed object class.

The data center AM changes the system level AM's policy profile based on their SLA violation status. In each HPC system AM, upon Timer1trigger event if there is SLA violation, the system AM will send an event message to the data center AM. In data center AM (PL7 and PL8) if still there is SLA violation (the condition of the policies) the data center AM changes the policy profile of the system based on whether they have an SLA violation (changes it to SLA base) or not (changes to Green). We expect that by changing policy profile of system dynamically we get better results in terms of total energy consumption and still limit the number of SLA violations.

The data center level AM has four policies: make a system *SLA* based or make it Green (policy profile). There is also a policy to deal with the Cooler - if the AM in the cooler detects a red temperature then that triggers an event in data center AM, and in response the data center AM “blocks” an HPC system for a period of time (SysBlockTimer block timer one of MOProperties of the data center class is set to 120) to relieve data center

load. What we have simulated for blocking HPC system is not running any jobs from the workload and in case of new arrival jobs just queuing them and not dispatching them to the compute nodes.

7.5.4 Experimental Scenarios

In this Scenario we consider five different experiments have been considered to evaluate the performance of having multiple autonomic managers with varying sets of policies.

Experiment 2.1. No management: The data center has the two running HPC systems, one with 30 compute nodes, one with 20 compute nodes..

Experiment 2.2 There is a manager at the HPC system level, which has an *SLA* policy profile (see Table 7-18). This scenario runs for the small HPC system of 20 compute nodes (we assume that the large HPC system is not running in the data center). The goal here is to evaluate the impact of the *SLA* policy profile on power and performance.

Experiment 2.3. There is a manager at the HPC system level, which has a *Green* policy profile i.e. Green (see Table 7-18). Again, this scenario runs for the small HPC system (the large HPC system is not running in the data center). The goal here is to evaluate the impact of the *Green* policy profile on power and performance.

Experiment 2.4. In this experiment we have two managers: one AM at the system level (the small HPC system is running) and the data center level. We are aim to evaluate the impact of dynamically changing policy profiles of the HPC system on the power and performance.

Experiment 2.5. In this experiment we have both HPC systems with their AMs running, an AM at the data center level and the cooler has its own manager that just checks for its maximum inlet temperature. If the inlet temperature is greater than the red temperature of hardware in the data center (specified in the data center configuration), the cooler AM sends a message (`UpdateHeartbeat` message) to the data center AM asking it to do something (refer to Table 7-18). In this experiment, the policy available to the AM in

the data center directly indicates that hpcSys1 should be "blocked" – essentially decrease processing by not executing additional jobs. Obviously, the blocked system will suffer from more SLA violations but the gain is that this decision addresses exceeding the red temperature for the whole data center. System priority is defined with the HPC system configuration.

7.5.5 Results of Experiments

The results of running these experiments are shown in Table 7-19. The first experiment does not have a management module and has two HPC systems. All compute nodes are running with their basic MIPS level. Running these systems under the workloads results in the inlet temperature of the cooler exceeding the red temperature 91 times; as a result, the simulator is not able to calculate the total energy consumed.

Experiment 2.2 and 2.3 involve a single HPC system with a manager. The *Green* policy profile consumes less energy and power than the same HPC system with the *SLA* policy profile while the number of SLA violations is about the same. This scenario shows how a small difference in policies can affect the overall behavior.

In experiment 2.4, we consider an AM at the data center level and its policy profile is *Green* (see Table 7-18). The data center AM with the *Green* policy profile is configured to dynamically change the policy profile of system level AMs in accordance with the system's SLA violations; if there are SLA violations at the system level, its policy profile is altered to be *SLA* based in order to put more priority on achieving SLAs than on energy conservation. The result shows that by having a data center level manager able to dynamically switch between a *Green* policy profile and an *SLA* policy profile, both energy conservation can be achieved while limiting the number of SLA violation.

Table 7-19. Comparison between different scenarios

Experiment	No management system (experiment	Single AM in system level (experiment 2.2 and 2.3)	Multiple AMs (data center level and system level) (experiment 2.4 and 2.5)

	2.1)				
Policy Profile	N/A	Green	SLA	DC AM Policy Profile is Green	
Num. of HPC systems in DC	2	1		1	2
Number of SLA Violations	448	189	187	189	454
Total energy consumption (Watt × Simulation Time)	N/A	8,000,000	9,800,000	8,318,000	23,171,143
Mean power consumption (Watt)	N/A	6,430	7,287	6,518	11,956
Number of times crossing red temperature	91	0	0	0	3
Number of exchanged messages	0	226	22	253	2,764

Table 7-19 summarizes the result of conducting different scenarios. The parameters are the number of SLA violations, total energy consumption and average power consumption during each experiment conducting time. Energy is calculated based on total power consumption multiply to total experiment conducting time (Watt × Sec= Joule). Another parameter is number of time the temperature of room is exceeded the red temperature. In total, we see that by policy based management can manage to decrease the energy consumption while maintaining the SLA level.

7.6 Scenario #3 Peer to Peer and Hierarchy AMs

In this scenario we look at a management strategy that relies on AMs as neighbors of each other. To do so, we configure one enterprise system with three applications hosted on it. The applications are different in terms of their computing intensity, the number of compute nodes and workloads. The original workload are from worldcup99 [20] which is web based requests with an average 9500 requests per second. In our experiment, we scaled up the workload for each of the applications. The scaling factor for each

application depends on the number of compute nodes assigned to the application and compute intensity of application.

This scenario also focuses on autonomic managers for an Enterprise system and for applications. The topology of managers is as follows: the manager at the level of the Enterprise system has priority over the managers for the applications; the application managers have the same priority, i.e. are peers.

Figure 7-9 shows the AM topology for in this scenario and focus of the policies for each AM. The AM at the system level elastically allocates/releases compute nodes from applications according to their need which is inferable from their SLA violation status and their compute node CPU frequency level. Moreover, the AM at the system level can change the policy profile of its application level AMs. The AMs attached to each application adjusts the CPU frequency of its compute nodes (if possible) and makes the compute nodes idle or active depending on SLA violations.

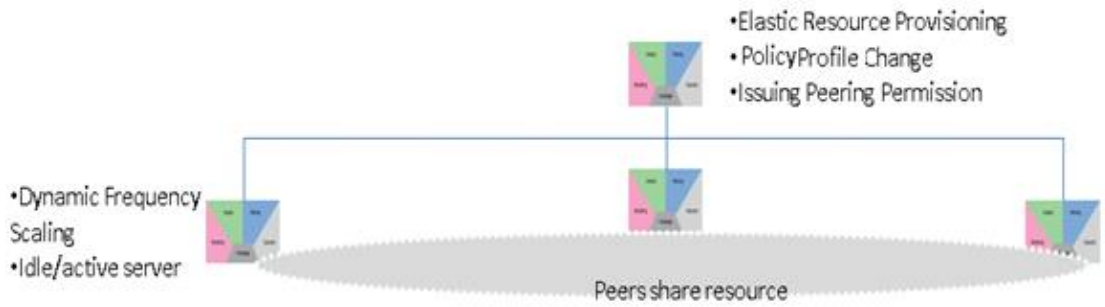

Figure 7-9. Topology of Managers and Policy Goals.

In this scenario, peers can also cooperate by releasing resources in favor of each other if and only if the AM at the system level lets them do that. The implementation of this idea is illustrated in the policy definitions. If the system level AM does not have available compute nodes for an application that has an SLA violation, it broadcasts a message to its peer applications requesting help for the violated application (UpdateHeartbeat). The message contains the buffer length of the application with the violation. Each application AM upon receiving the message, checks to see if its status (in terms of number of jobs in

their queue) is better than the violated application and if so they will release a node, otherwise they will do nothing.

Following tables shows the configuration of data center and its set of manageable objects.

Table 7-20. Data Center Set of Manageable Objects

dataCenterT	**SysList**={ Ent_Sys1} **Racks**={RackT} **RackT**={r1,r2,r3,r4,r5,r6,r7,r8,r9,10} r1 = {ch11,ch12,ch13,ch14,ch15} r2 = {ch21,ch22,ch23,ch24,ch25} r3 = {ch31,ch32,ch33,ch34,ch35} r4 = {ch41,ch42,ch43,ch44,ch45} r5 = {ch51,ch52,ch53,ch54,ch55} r6 = {ch61,ch62,ch63,ch64,ch65} r7 = {ch71,ch72,ch73,ch74,ch75} r8 = {ch81,ch82,ch83,ch84,ch85} r9 = {ch91,ch92,ch93,ch94,ch95} r10 = {ch101,ch102,ch103,ch104,ch105} ch11= {n11} ch12= {n12} ch13= {n13 } ch14= {n14} ch15= {n15 } ch21= {n21 } ch22= {n22 } ch23= {n23 } ch24= {n24 } ch25= {n25 } ch31= {n31 } ch32= {n32 } ch33= {n33 } ch34= {n34} ch35= {n35 } ch41= {n41 } ch42= {n42 } ch43= {n43 } ch45= {n45 } ch51= {n51 } ch52= {n52 } ch53= {n53} ch54= {n54} ch55= {n55}

	ch61= {n61 } ch62= {n62} ch63= {n63} ch64= {n64} ch65= {n65} ch71= {n71} ch72= {n72} ch73= {n73} ch74= {n74} ch75= {n75} ch81= {n81} ch82= {n82} ch83= {n83} ch84= {n84} ch85= {n85} ch91= {n91} ch92= {n92} ch93= {n93} ch94= {n94} ch95= {n95} ch101= {n101} ch102= {n102} ch103= {n103} ch104= {n104} ch105= {n105} where $n_{i,}$ is a nodeT **ThermalModel** =< coolerList, RedTemp, ThermalMap > = <{coolerT}, 35, DMatrix>
nodeT	**Id** **MIPS**={1 1.04 1.4 } **FullLoadPowerConsumption**= {300, 336, 448} **ZeroLoadPowerConsumption**={100, 100, 128} **StandbyPower**= 5
coolerT	**CoP_Equ=** $0.0068*T^2+0.0008*T+0.458$

Table 7-21 illustrates configuration parameters of manageable objects for this scenario.

Table 7-21. Manageable Object definition for Scenario#3

Manageable Object

Ent_System1	**Type**=Ent_Sys1 **SysWorkload**={Ent_Application1, Ent_Application2, Ent_Application3} **Queue** **ResManagers**= {MHR) } **Sched**= {(FCFS)} **Racks**= {r1,r2,r3,r4}
Ent_Application1	**Id=1** **startTime**=1 **computeNodes** :<2,10> **computeIntensity**=(1,1000) **workload**=WorkLoad1 **SLA**= (2, 90%,60s) terminationTime=0
Ent_Application2	**Id=2** **startTime**=1 **computeNodes** :<2,10> **computeIntensity**=(2,1000) **workload**=WorkLoad2 **SLA**= (2, 90%,60s) terminationTime=0
Ent_Application3	**Id=3** **startTime**=1 **computeNodes** :<3,15> **computeIntensity**=(3,1000) **workload**=WorkLoad3 **SLA**= (2, 90%,60s) terminationTime=0

Configuration of managed object classes and their associated sensors, events and actuators are illustrated in Table 7-22. The enterprise system object class has an SLAViolation event corresponding to each application which are triggered when the application has SLA violation. For its sensors, we have defined compPwrPercent (explained in the following) for each application and a queue length sensor for each applications (queueLengt(applicationId)). The system level AM has an SLA violation sensor for each application. The system level AM has a sensor for the response time of an applications, i.e. reponseTimeInLastEpoch(applicationId) and also has a sensor idleNode(id) that returns number of idle nodes in the system.

The actuators for the enterprise system object class include allocating a node to an application (allocateNode(application)) and, releasing a node from an application (releaseANode(applicationID)).

For the enterprise applications, we have two events: triggerTimer1 (i.e. defined as its MOProperties) and CooperationEvent. CooperationEvent is triggered by the system level AM in the management system. When, the system level AM wants to allocate a node to an application in need and there is no node available to allocate, the system level AM needs to notify all applications inside the system. This is done via an event trigger in our management system. As explained in section 6.6, the system level AM needs to send an UpdateHeartbeat message with EventCode to all of its application AMs. This event (message) is named as CooperationEvent in application managers.

There are also actuators for then enterprise application class (EntAppClass) which their descriptions are as follows:

- increaseFreqFullyUtilized(id): which increases the frequency of all fully utilized nodes of the application with given id (if possible).
- activateAllSleepNodes(id): which activates all sleeping nodes of application with given id.
- activateHalfSleepNodes(id): which activates half of the sleeping nodes of application with given id.
- increaseFreqBusyNodes(id): which increases the frequency of all busy nodes of the application with given id.
- decreaseFreqAllBusyNodes(id): in which decrease frequency of all busy nodes of the application with given id.
- sleepAllidleNodes(id): which puts to sleep all idle nodes of the application with given id.

Upon arrival of any application in the data center it gets its own id and this id will be sent to the AM as configuration parameter i.e. explained in Chapter 6 Algorithm 5. Table 21 illustrates the MO classes in this scenario. EntAppClass has Ent_Application as its

managed object. Our set of manageable objects and the managed object classes are defined in Table 7-23 and Table 7-24.

Table 7-22. Configuration Managed Object Class Scenario#3

Managed Object Class	
EntSysClass	**MOs=** {Ent_Sys1} **Event**={SLAViolation1, SLAViolation2,SLAViolation3} **Sensor**={ compPwrPercent(id), queueLengt(id), reponseTimeInLastEpoch(id), idleNode(id)} **Actuator**={ allocateNode(applicationID), releaseANode(applicationID)} **MOProperties**={}
EntAppclass	**MOs=** {Ent_Application} **Event= {**CooperationEvent, trigger Timer1} **Sensor**={ reponseTimeInLastEpoch(id) , queueLength(id)} **Actuator**={ increaseFreqFullyUtilized(id), activateAllSleepNodes(id), activateHlafSleepNodes(id),increaseFreqBusyNodes(id), decreaseFreqAllBusyNodes(id), sleepAllidleNodes(id), releaseAnIdleNode(applicationID)} **MOProperties**={ (Timer1 Value,60s)}

Table 7-23. Data Center Set Of Manageable Object.

Set of Manageable object
MO_{DC}={dataCenterT}
MO_{Racks}={ rackT}
MO_{node}= {nodeT}
$MO_{EntrpriseSys}$= { Ent_System1}
MO_{Apps}= { Ent_Application1, Ent_Application2, Ent_Application3}

Table 7-24. Data Center Managed Object Class.

Managed Object Class
EntSysClass
EntAppClass
NodeClass

AM classes, their associated MO classes, and AM pair and topology of management system are illustrated in following tables. As mentioned in earlier in Chapter 6. (Algorithm 5) our model if flexible to determine which AM should manage the current new arrival MO, but in our experiments without losing generality, we assume that each AM has only one MO in its list of MO. So, in this experiment we assume that we have 3 different AM object for three different enterprise applications.

Table 7-25. AM Classes and MO Classes

AM Class	Associated MO class
AMEntSys	EntSysClass, EntAppClass
AMEntApp	EntAppClass, NodeClass

SLAViolationCheck is a function that checks if 90% of the response in the last reponseTimeInLastEpoch is less than 2 seconds and returns 1 otherwise returns 0 for application with given id. Note that reponseTimeInLastEpoch is defined in the list of EntApp1Class, EntApp2Class, and EntApp3Class sensors.

Table 7-26. AM Pair and Management Topology

AMPair	(AMEntSys, AMEntApp ,1), (AMEntApp, AMEntApp,0)
T_{DC}	V={ AMEntSys, AMEntApp } E=AMPair

Table 7-27. Policy definition Scenario#3

SLA Policy Profile For AM Attached to AMEntApp	PL0:
	On Event: **Timer1trigger**
	If (SLAViolationCheck (90%,reponseTimeInLastEpoch,2,id)>0)
	begin
	increaseFreqBusyNodes(id) → activateAllSleepNodes(id) | activateAllSleepNodes(id)
	End

Green Policy Profile For AM Attached to $EntApp_{id}$	PL1:
	On Event: **Timer1trigger**
	If (SLAViolationCheck(90%,reponseTimeInLastEpoch,2,id)>0)
	begin
	increaseFreqFullyUtilized(id)→activateHalfSleepNodes(id) |activateHalfSleepNodes(id)
	end
	PL2:
	On Event: **Timer1trigger**
	If (SLAViolationCheck(90%,reponseTimeInLastEpoch,2,id)<0)
	Begin
	decreaseFreqAllBusyNodes(id) → sleepAllidleNodes(id)| sleepAllidleNodes(id)
	end
	
	PL3:
	On Event: CooperationEvent
	If (queueLength(id)<ReceiveMessage.getInfo())
	begin
	releaseAnIdleNode(id)

	end

AMEntSys Policy Set	PL4[11]:
	On Event: (**SLAViolation$_1$ | SLAViolation$_2$ |SLAViolation$_3$**)
	if (SLAViolationCheck(90%,reponseTimeInLastEpoch(1),2,1)>0)
	/*SLA$_1$ is violated*/
	begin
	Send message (ChangePolicyProfile , AMEntApp1, SLA based)
	end
	else
	begin
	Send message (ChangePolicyProfile , AMEntApp1,Green)
	end
	
	PL5:
	On Event: (**SLAViolation$_1$ | SLAViolation$_2$|SLAViolation$_3$**)
	if (SLAViolationCheck(90%,reponseTimeInLastEpoch(2),2,2)>0)
	/*SLA$_2$ is violated*/
	begin
	Send message (ChangePolicyProfile , AMEntApp2, SLA based)
	end
	else
	begin
	Send message (ChangePolicyProfile , AMEntApp2,Green)
	end
	
	PL6:
	On Event: (**SLAViolation$_1$ | SLAViolation$_2$|SLAViolation$_3$**)
	if (SLAViolationCheck(90%,reponseTimeInLastEpoch(3),2,3)>0)
	/*SLA$_3$ is violated*/
	begin
	Send message (ChangePolicyProfile , AMEntApp3, SLA based)
	end
	else
	begin
	Send message (ChangePolicyProfile , AMEntApp3,Green)
	end
	
	PL7:
	On Event: (**SLAViolation$_{id}$**)
	if (compPwrPercent(id)>0.5 & idleNode() >0)
	begin

[11] The idea in policyPL4, PL5,and PL6 is that upon any application SLA violation, the violated application policy profile is changed to SLA based and rest changed to Green.

	allocateANode(id)
	end
	
	PL8:
	On Event: (**SLAViolation$_K$|SLAViolation$_L$**)
	if (compPwrPercent(J) <0.5)
	begin
	releaseANode(J)
	end
	
	PL9:
	On Event: (**SLAViolation$_{id}$**)
	if (compPwrPercent(id)>0.5 & idleNode() = 0)
	begin
	Send message (UpdateHeartbeat, Boradcast, queueLength(id), EventCode)
	end

EntSysAM PolicyProfile1	PL7,PL8
EntSysAM PolicyProfile2	PL4,PL5,PL6, PL7,PL8
EntSysAM PolicyProfile3	PL4,PL5,PL6, PL7,PL8,PL9

Note that an enterprise system has SLA sensors for its applications in its sensors list.

Four different cases have been conducted and compared in this scenario:

- *Experiment 3.1.* The system level AM uses profile policy PolicyProfile1 and all applications make use of the policy profile that is SLA based. Briefly, the system level AM just allocates/releases resources to/from an application according to their SLA violation status. The application level AM adjusts the frequency level of application's compute nodes.
- *Experiment 3.2.* The system level AM uses policy profile PolicyProfile1 and all applications use the policy profile Green. The system level AM just allocates/releases resource to/from an application according to their SLA violation status. The application level AM adjusts the frequency level of application's compute nodes. This experiment is identical to Experiment 3.1 except that the policy profile used the application AMs make use the Green profile instead of an SLA based one.

- *Experiment 3.3.* The system level AM profile policy is PolicyProfile2, which changes the profile policy of all of its applications based on their SLA violation status. If the application has an SLA violation, then the system level AM changes the application's profile policy to the SLA based policy profile, otherwise it will be changed to Green policy profile. Initially, each application AM starts with the Green policy profile. Moreover, it allocates/releases resources to/from applications according to their SLA violation status. The application level AM adjusts the frequency level of application's compute nodes.
- *Experiment 3.4.* The system level AM policy profile is PolicyProfile3. In this situation, the system level AM changes resources and the policy profile of its application level AMs (according to the status of the application SLA violations, the application policy profile will be changed to an SLA based or Green policy). The only difference here is that if the system level AM does not have an available node to allocate, it will broadcast a message to all other AMs at the application level to try to obtain resources (compute nodes) for the application with the SLA violation. An application AM upon receiving the message will compare their queue length with the queue length of the AM with the SLA violation. If they are in better shape in terms of queue length, they will release a node so that the system level AM can allocate that released node to the AM with the violation (through resource manager in system level). After the node is released, since the system level AM has a new node available to allocate, the system level AM will use this node.

Release/Allocation of compute node to the application in this scenario is straightforward. We just check how busy are the nodes allocated to the application are. To do so, we need to know if there is room for frequency scaling (if the CPU supports that) and in which frequency level their CPUs are. If all of them are at full CPU frequency then this application cannot run more effectively, and the application needs to be allocated to a new node. The parameter compPwrPercent is an indicator, shows how much the applications (with given id) utilizes its compute nodes frequency scaling. An AM at the

level of an application calculates this; compPwrPercent is one of system level AM sensors. Upon initialization of an application, the resource manager at the system level allocates the minimum requested compute nodes to the application and in each epoch time according to SLA violation of the application and its compPwrPercent level; it will allocate/release nodes to/from the application.

Table 7-28. Comparison between different scenarios

Profile Policy System Level	Experiment 3.1 PolicyProfile1	Experiment 3.2 PolicyProfile1	Experiment 3.3 PolicyProfile2	Experiment 3.4 PolicyProfile3
Profile Policy Application Level	SLA	Green	Initial: Green	
Num. of SLA violation in entApp1	3	114	17	68
Num. of SLA violation in entApp2	6	12534	8070	4020
Num. of SLA violation in entApp3	31	873	1062	770
total energy consumption (Watt * Simulation Time)	2.9*10^8	1.95*10^8	2*10^8	2.08*10^8
Mean power consumption (Watt)	4451	2976	3113	3180

In Table 7-28, we illustrate the result of running these different scenarios. As shown, aside from the better performance with the SLA policy profile compared to the Green policy profile in terms of SLA violation, we see that making the application level AMs share resources with each other (policy profile 3) results in fewer total SLA violations and the power consumption has not changed much.

Chapter 8

Conclusion and Future Work

Administrator independent and autonomous management of systems in a data centers is the central focus of this research. In this work, a policy based management system has been developed for managing a data center and its systems with a focus on managing data center energy consumption. Considering the inherent nature of data centers, which are heterogeneous, complex, distributed and dynamic the management system needs to be flexible enough to be applicable for this environment. Our main assumption here is that we can use all available approaches and techniques for energy and power management that have been proposed in the literature.

The main contributions of this research are:

- A general management model has been developed which lets the administrator define the management model by defining managed objects, autonomic managers, and the topology of interaction among the managers. The model also captures the definition and creation of policies for each class of autonomic managers. In this model, the administrator can change the behavior of the management system by defining new managed objects, new managers and even defining new sets of policies.

- The notion of a management topology has been introduced which can be used to model the communication among autonomic managers within a management system. There is no limitation on the granularity of managers or connection between managers in the data center.

- The communication between managers is a loosely coupled communication and is done through message passing. A set of primary message types for manager communication has been defined.

- Given the configuration of a management system and the definition of managers, a number of algorithms for the deployment of an instance of a management system have been introduced.

- The proposed management system has been evaluated with number of scenarios and has shown promising results; which means that with management system we can get better energy consumption and reduced impact on the performance of the data center.

- To develop the scenarios, we have implemented a data center simulator which simulates the data center behavior under different types of systems and workloads. The simulator is able to compute the energy consumption of the data center as well. We can define different type of hardware (servers, chassis, and racks) and applications. The simulator is scalable which means that we can increase the number of nodes. The simulator is the secondary outcome of this research.

- Another path for future work on improving the flexibility and generality of the data center simulator:

    - It should be able to support a number of "common" data center topologies, e.g. fat trees;

    - As the trend of virtualization is going toward containers, the simulator needs to support containers within systems;

    - The current thermal model used in the simulator does not consider humidity as one of the factors in calculating the power consumption of data center. One of the goals for future is working on comprehensive thermal model that is closed to the thermal behavior of a data center. Refer to that proposed in [61];

    - Given that alternative energy sources are becoming common, it would be useful to include a power model in the simulator that accounted for the use

of renewable energy sources and how they could be taken into account in the management system.

To fully leverage, the notion of an autonomous management system there are additional areas of research that are needed:

1. Conflicts in manager decisions can originate in the action of their policies. So, whenever the administrator defines the policy sets for managed objects and managers, there needs to be a sanity phase which checks for the potential conflict.

2. The overhead of message passing and communication with policy repository is a potential issue in the management system. This could be addressed by defining management for the management system. This means that management system could have manager that monitors the heartbeat of management system (in which the number of messages could be an indicator) and based on that makes decision regarding any potential changes in the management system.

3. Part of the management system could be imbedded in the application context and as a management layers being part of application protocol stack.

4. The collaboration between managers could evolve based on how the system and policies change; further work on how policies can be used for this needs to be explored.

5. Evolving the management system algorithms for dealing with sudden changes in the management system e.g. disconnection of a manager and security.

6. Evolving the simulator to include more automated means of specifying systems, policies, etc. and creating a more sophisticated user interface.

The proposed autonomic management system has vast application in management of cloud environment and distributed data center due to its administrator independent and policy based attributes.

References

[1] Koomey, Jonathan. "Growth in data center electricity use 2005 to 2010." A report by Analytical Press, completed at the request of The New York Times (2011): 9.

[2] Forrest, William, James M. Kaplan, and Noah Kindler. "Data centers: how to cut carbon emissions and costs." *McKinsey on business technology* 14.6 (2008): 4-13.

[3] Google green, http://www.google.com/green, Last visited Jun.2015.

[4] Khargharia, Bithika, Salim Hariri, and Mazin S. Yousif. "Autonomic power and performance management for computing systems." *Cluster computing* 11.2 (2008): 167-181.

[5] Anthony, Richard John, Mariusz Pelc, and Haffiz Shuaib. "The interoperability challenge for autonomi c computing." (2011): 13-19.

[6] Ahmad, Faraz, and T. N. Vijaykumar. "Joint optimization of idle and cooling power in data centers while maintaining response time." *ACM Sigplan Notices*. Vol. 45. No. 3. ACM, 2010.): 1-20.

[7] Gupta, Sandeep KS, et al. "Gdcsim: A simulator for green data center design and analysis." *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 24.1 (2014): 37-53.

[8] Kennedy, Catriona. "Decentralized meta-cognition in context-aware autonomic systems: Some key challenges." *American Institute of Aeronautics and Astronautics (AIAA) AAAI-10 Workshop on Metacognition for Robust Social Systems, Atlanta, Georgia*. 2010): 134-149.

[9] Quitadamo, Raffaele, and Franco Zambonelli. "Autonomic communication services: a new challenge for software agents." *Autonomous Agents and Multi-Agent Systems* 17.3 (2008): 457-475.

[10] Ngolah, Cyprian Foinjong, and Behrouz H. Far. "A tutorial on agent communication and knowledge sharing." *University of Calgary, SENG609* 22.

[11] Nisan, Noam, ed. *Algorithmic game theory*. Cambridge University Press, 2007.

[12] Boutaba, Raouf, and Issam Aib. "Policy-based management: A historical perspective." *Journal of Network and Systems Management* 15.4 (2007): 447-480.

[13] Kephart, Jeffrey O., and David M. Chess. "The vision of autonomic computing." *Computer* 36.1 (2003): 41-50.

[14] https://msdn.microsoft.com/en-us/library/bb833022.aspx, Last visited June 2015.

[15] http://institutes.lanl.gov/hec-fsio/ Last visited July.2014

[16] Mukherjee, Tridib, et al. "Spatio-temporal thermal-aware job scheduling to minimize energy consumption in virtualized heterogeneous data centers." *Computer Networks* 53.17 (2009): 2888-2904.

[17] Chen, Yiyu, et al. "Managing server energy and operational costs in hosting centers." *ACM SIGMETRICS Performance Evaluation Review*. Vol. 33. No. 1. ACM, 2005: 303-314.

[18] Mukherjee, Tridib, et al. "Model-driven coordinated management of data centers." *Computer Networks* 54.16 (2010): 2869-2886.

[19] Kephart, Jeffrey O., et al. "Coordinating Multiple Autonomic Managers to Achieve Specified Power-Performance Tradeoffs." International *Conference* on Autonomic Computing. Vol. 7. 2007: 24-32.

[20] World cup traffic log: http://ita.ee.lbl.gov/html/contrib/WorldCup.html, Last visited June 2015.

[21] Moore, Justin D., et al. "Making Scheduling" Cool": Temperature-Aware Workload Placement in Data Centers." *USENIX annual technical conference, General Track*. 2005: 61-75.

[22] IBM, "An architectural blueprint for autonomic computing, 4th edition." http://www-01.ibm.com/software/tivoli/autonomic /pdfs/AC_Blueprint_White_Paper_4th.pdf, Last visit June 2015.

[23] Al-Shishtawy, Ahmad. "Enabling and Achieving Self-Management for Large Scale Distributed Systems: Platform and Design Methodology for Self-Management." (2010). Last Visit June 2015. [Online] Available: https://www.sics.se/~ahmad/phdthesis/AhmadAlShishtawy-Licentiate.pdf

[24] Padala, Pradeep, et al. "Performance evaluation of virtualization technologies for server consolidation." *HP Labs Tec. Report* (2007).

[25] Meisner, David, Brian T. Gold, and Thomas F. Wenisch. "PowerNap: eliminating server idle power." *ACM Sigplan Notices*. Vol. 44. No. 3. ACM, 2009: 205-216.

[26] Ranganathan, Parthasarathy, et al. "Ensemble-level power management for dense blade servers." *ACM SIGARCH Computer Architecture News*. Vol. 34. No. 2. IEEE Computer Society, 2006: 66-77.

[27] Lorch, Jacob R., and Alan Jay Smith. "Improving dynamic voltage scaling algorithms with PACE." *SIGMETRICS/Performance*. 2001.): 1-9.

[28] Heath, Taliver, et al. "Mercury and freon: temperature emulation and management for server systems." *ACM SIGARCH Computer Architecture News*. Vol. 34. No. 5. ACM, 2006: 106-116.

[29] https://www.facebook.com/notes/facebook-engineering/designing-a-very-efficient-data-center/10150148003778920, Last visited June 2015.

[30] Belady, Christian, et al. "Green grid data center power efficiency metrics: PUE and DCiE." The green grid (2008): 1-9.

[31] Wang, Lizhe, et al. "Towards thermal aware workload scheduling in a data center." *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on*. IEEE, 2009: 116-122.

[32] Abbasi, Zahra, Georgios Varsamopoulos, and Sandeep KS Gupta. "Thermal aware server provisioning and workload distribution for internet data centers." *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, 2010: 130-141.

[33] VMware Inc., How VMware virtualization right-sizes IT infrastructure to reduce power consumption, 2009. Last visited June 2015. [Online]. Available: http://www.vmware.com/files/pdf/WhitePaper_ReducePowerConsumption.pdf

[34] Chase, Jeffrey S., et al. "Managing energy and server resources in hosting centers." *ACM SIGOPS Operating Systems Review*. Vol. 35. No. 5. ACM, 2001,103-116.

[35] Garg, Saurabh Kumar, et al. "Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers." *Journal of Parallel and Distributed Computing* 71.6 (2011): 732-749.

[36] Abbasi, Zahra, Madhurima Pore, and Sandeep KS Gupta. "Impact of workload and renewable prediction on the value of geographical workload management." *Energy-Efficient Data Centers*. Springer Berlin Heidelberg, 2014. 1-15.

[37] G. Demasi, "More renewable energy for our data centers," Last visited June 2015. [Online]. Available: http://googleblog.blogspot.com.ar/2012/09/more-renewable-energy-for-our-data.html

[38] R. Mcmillan, "Apple vows to build 100% renewable energy data center," http://www.wired.com/wiredenterprise/2012/04/applerenewable/, Last visited June 2015.

[39] Kusic, Dara, et al. "Power and performance management of virtualized computing environments via lookahead control." *Cluster computing* 12.1 (2009): 1-15.

[40] Al-Shishtawy, Ahmad, et al. "A design methodology for self-management in distributed environments." *Computational Science and Engineering, 2009. CSE'09. International Conference on*. Vol. 1. IEEE, 2009: 430-436.

[41] Schneeweiß, Christoph. *Distributed decision making*. Springer, 2003.

[42] Röblitz, Thomas, et al. "Autonomic management of large clusters and their integration into the grid." *Journal of Grid computing* 2.3 (2004): 247-260.

[43] http://eu-datagrid.web.cern.ch/eu-datagrid/, Last visited June 2015.

[44] Pore, Madhurima, et al. "Techniques to Achieve Energy Proportionality in Data Centers: A Survey." *Handbook on Data Centers*. Springer New York, 2015. 109-162.

[45] https://github.com/fnorouz/simulator/ Last visited January 2015.

[46] Baldassari, James D., et al. "Autonomic cluster management system (ACMS): A demonstration of autonomic principles at work." *Engineering of Computer-Based Systems, 2005. ECBS'05. 12th IEEE International Conference and Workshops on the*. IEEE, 2005: 512-518.

[47] http://www.ponder2.net/. Last visited July.2014

[48] Corkill, Daniel D. "Blackboard systems." *AI expert* 6.9 (1991): 40-47.

[49] Smith, Reid G. "The contract net protocol: High-level communication and control in a distributed problem solver." *Computers, IEEE Transactions on* 100.12 (1980): 1104-1113.

[50] Chen, Gong, et al. "Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services." *NSDI*. Vol. 8. 2008: 337-350.

[51] Tang, Qinghui, Sandeep Kumar S. Gupta, and Georgios Varsamopoulos. "Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach." *Parallel and Distributed Systems, IEEE Transactions on* 19.11 (2008): 1458-1472.

[52] Tesauro, Gerald, et al. "A multi-agent systems approach to autonomic computing." *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*. IEEE Computer Society, 2004: 464-471.

[53] Das, Rajarshi, et al. "Autonomic multi-agent management of power and performance in data centers." *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*.

International Foundation for Autonomous Agents and Multiagent Systems, 2008: 107-114.

[54] Bahat, Raphael M., et al. "Using policies to drive autonomic management." *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*. IEEE Computer Society, 2006: 475-479.

[55] http://impact.asu.edu/BlueTool/wiki/index.php/BlueSim, Last visited June 2015.

[56] Dong, Yi, et al. "Improving Energy Efficiency for Mobile Media Cloud via Virtual Machine Consolidation." *Mobile Networks and Applications* (2015): 1-10.

[57] http://www.cloudbus.org/cloudsim/ Last visited June 2015.

[58] https://impact.asu.edu/publication/GDCSim.pdf Last visited June 2015.

[59] https://www.eucalyptus.com/ Last visited June 2015.

[60] Keller, Gastón, et al. "A hierarchical, topology-aware approach to dynamic data centre management." Network Operations and Management Symposium (NOMS), 2014 IEEE. IEEE, 2014.

[61] Manousakis, Ioannis, et al. "Environmental conditions and disk reliability in free-cooled datacenters." Proceedings of the 14th Usenix Conference on File and Storage Technologies. USENIX Association, 2016.

Appendix A: Overview of Simulator

Simulation versus real experiments. Running a data center (with compute nodes and cooling equipment together) which is capable of supporting different configurations of applications and different types of hardware is the preliminary step. Simulation of such environment, rather than working with a small, toy data center, is the best solution for our purpose for several reasons. First, running a real data center, even a small one, is costly because providing a thermal insulation environment with chillers and several servers in it is expensive. Second, evaluation of different management ideas need to be evaluated in different types of computing loads on several hundreds of compute nodes, possibly heterogeneous, which may be hard to attain in a real data center. Third, examining a specific policy for different data center layouts (different placement of chillers and compute nodes or different cold/hot aisle plan) is more feasible with simulation. A simulator opens up the possibility of evaluating different scenarios; this is most important at this stage.

Other simulators. Several other data center simulators have been proposed and exist. CLOUDSIM tries to simulate the dynamic behavior of applications on virtual machines running on a cloud [57]. This simulator has powerful modules for evaluating virtualization policies. An energy management module for CLOUDSIM has been developed and they now take into account the computing power of the host running the virtual machine workload; they do not take into consideration the cooling power consumption which obviously gets affected by the physical layouts of the data center.

GDCSIM [58] is another simulator which provides simulation of the thermal behavior of a data center with a given physical layout. GDCSIM does not provide a framework that can support different types of applications/users running jobs nor does it provide a mechanism for considering different SLA.

Key Features. The first objective behind the design of the simulator was the development of a platform for evaluation of different arrangement of autonomic computing agents across the abstract model of data center and to explore the influence of the granularity of managers on minimizing energy consumption of a data center with

compliance to SLAs. To the best of our knowledge, this simulator is the only simulator which tries to simulate the management of energy consumption of a data center inspired by autonomic computing. The thermal models used in the simulator are validated model based on the BlueSim thermal data center simulator [55]. Another key feature of the simulator is that it supports different types of computing systems (enterprise, interactive and high performance computing) and different types of workloads with various SLA parameter definitions. Likewise, its hierarchical abstraction model aims to accommodate monitoring and management modules.

Key challenges. CPU utilization is assumed to be the primary parameter in determining energy consumption of a server. This is a general assumption in the literature and so our focus has been on the timely calculation of server CPU utilization while jobs are running. Dependency of this simulator on BlueSim (to get thermal model) can also be considered a challenge since such a model is needed, although without having thermal model of the data centre computing power calculations and other features can still be done.

A.1. Overview

The initial objective in designing this simulator was to develop a platform to evaluate different collaboration models between different autonomic computing agents and to explore the influence of different collaboration architectures on energy consumption of a data center while considering compliance with SLAs. The simulator is basically an infrastructure for this research. The novelty of this approach lies in considering the datacenter as an entity managed by autonomic agents. In our case, we assume that these agents make use of defined policies, often defined at various levels, in order to manage a variety of aspects of a data center, from allocation of jobs to management or workload.

A.1.1 General features

Given our objectives, the simulator needed to support a number of features and capabilities:

- Support heterogeneous servers: the ability to support different types of servers with different levels of energy consumption and computing power.
- Support different workload models: the ability to define several types of workloads, e.g., compute intensive or interactive jobs.
- Support dynamic resource allocation for dynamically allocating server to different meet application requirements under changing workloads.
- Support different workload scheduling algorithms.
- Support different types of SLAs: the ability to define different kinds of SLA for each type of workload and also keep tracking of SLA violations.
- The ability to record energy consumption of the data center (computing and cooling).
- The ability to change CPU working frequency of servers.

The main assumption in our simulator is that the *thermal model* of the data center is given.

A.1.2 Architecture

Figure 2 shows the overall structure of the simulator. The simulator gets the thermal profile of the data center and simulator inputs, e.g. configuration of systems and computing loads. During the simulator run, the results are output to a logfile which includes data on the thermal behavior of the system and SLA violations.

The front-end configures the simulator and records outputs. The back-end runs the workload and simulates the system. Dynamic changes in the system configuration based on policies is done in the autonomic manager module, which may contain several modules, e.g. representing several layers autonomic mangers The thermal profile of the data center is extracted from the BlueSim package. The basic objective of BlueSim is to generate a Heat Recirculation Matrix (HRM) for a given topology of data center.

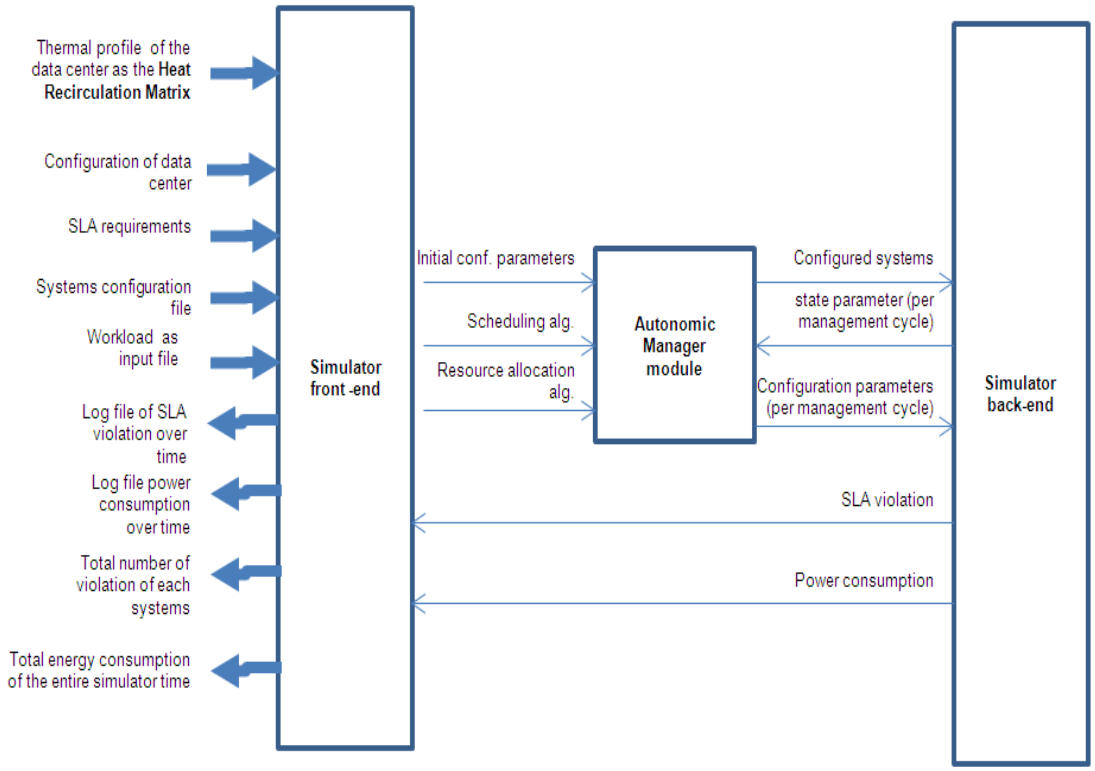

Figure 2. Overall structure of the simulator.

A.2. The Simulator Operation

The main processes in the simulator after the configuration step, are running the workload, resource management, and timely calculation of energy consumption. Energy consumption is calculated using the thermal model which is defined in the configuration module (configurePysicalLayout). Overall pseudo code of the simulator is shown in following table.

Table 1. The simulator lifecycle pseudo code

Data center life cycle procedure

```
procedure DClifeCycle()
begin
    dataCenterInitialization();
    while(jobs are not finished)
      begin
          for each systems  i in the data center
                    SysList.get(i).RunACycle();
```

```
procedure System ::RunACycle()
begin
          for each application app in the system
                    app.runAcycle();
          end for
```

```
            end for
            Compute power consumption();

            if( managerIsAttached)
                  runManagementCycle();
            end if

      end while
end procedure
```

```
procedure dataCenterInitialization ()
begin
     configurePysicalLayout();
     configureAMTopology()
     for  all number of requested system
     begin
            Read confing file();
            initialize systems();
            //insert current system in data center system list
            SysList.add(sys) ;
    end for
end procedure
```

```
            if( managerIsAttached)
                  runManagementCycle();
            end if

end procedure

procedure application::RunACycle()
begin
      readJob();
      dispatchJobToComputeNode();
      for each computeNode node in the application
            node.calculateCPUutilization();
      end for
      check&SetViolation();
      if managerIsAttached()
            runManagementCycle();
      end if
end procedure
```

For each system there is an output file which records each time a SLA violation occurs and the information recorded depends on type of system:

For interactive and enterprise applications, each time an SLA is violated, based on the SLA definition for an application, the violation is recorded in output file of corresponding system.

Finally, another output file will record power consumption of whole data center during simulation time. Each record of this file includes: *computing power*, *cooling power*, and *time*.

A.3. Evolving the simulator for new data center

Now, the question is that what would be the action plan for using the simulator for a new data center?

- Define the physical configuration of data center .
- Get a thermal map for given data center using BlueSim as described in the section on the thermal map.
- Define the systems, applications, SLAs, and workloads.
- Defining the position of autonomic managers and management scenarios: to do so, programming is needed.

- Developing management policies and scenarios.

A.4. Extracting thermal map

The Arizona State datacenter team introduced a package -BlueSim- which aims to generate HRM for any given description of datacenter. BlueSim is a simulation package, which integrates various software for geometry generation, CFD simulation, and post processing. The main idea behind BlueSim is to generate an Heat Recirculation Matrix (HRM) for different configurations of the data centers. There are three main subcomponents in BlueSim:

- **Pre-processing**: The input to BlueSim is a high level XML-based data center description in Computer Infrastructure Engineering Language (CIELA). It has a range of elements that capture the generic layout of a data center: (i) equipment configuration, i.e. how is the arrangement of servers into chassis and chassis into racks, likewise the arrangement of these racks into rows (ii) physical datacenter layout, i.e. presence of raised floors, lowered ceilings. A parser parses the given XML specification to create a geometry file which depicts the layout of datacenter. The layout is then converted to a mesh file using GMSH, a standalone open source meshing software. *Note that for getting thermal model of a new data center, just defining the physical description of data center in CIELA is needed, the rest is done by BlueSim.*

- **Processing**: This component runs a series of offline CFD simulation of the specified data center to enable determination of HRM. The simulation scenarios aim to extract the effect of each chassis on others by fully utilizing the node and getting thermal information of other nodes from the CFD model.

- **Post-processing**: This component generates an array of different HRMs with different active server sets. Having number of HRMs based on different situations gives better thermal model precision. The current version generates only a single HRM with all the servers active. The claim they have plan for future releases of BlueSim that will generate an array of HRMs for different active server sets.

Appendix B: Examples of Workloads for Experiments

B.1 Sample HPC type workload used in Scenario #2.

Arrival Time	Duration	CPU estimation	# of requested CPU	Deadline
1	1	41.07	2	1
1	1	51.19	2	1
1	1	48.33	1	1
1	8	45.16	1	1
1	9	52.78	1	1
1	7	57.89	3	1
1	13	48.89	3	1
1	6	48	3	1
1	7	38.46	3	1
1	8	40.63	3	1
2	370	27.72	2	1
4	8	58.46	2	1
4	17	37.5	2	1
4	1	49.23	2	1
4	1	50	2	1
4	8	53.08	3	1
4	7	40.74	1	1
4	411	10.9	1	1
5	10	35.8	1	1
5	9	56.06	1	1
5	12	40.21	1	1
5	12	40.63	1	1
5	8	44.44	3	1
5	7	49.77	3	1
5	8	45.31	3	1
5	460	39.74	3	1

B.2 Sample of Enterprise workload used in Scenarios #1, #3.

Arrival Time	# of request
1	225
2	272
3	241
4	215
5	249
6	240
7	266
8	265
9	269
10	262
11	241
12	288
13	246
14	267
15	304
16	223
17	243
18	266
19	222
20	255

Curriculum Vitae

Name: Forough Norouzi

Post-secondary Education and Degrees:

Shiraz University
Shiraz,Fars,Iran
1995-1999 Engineering.

Sharif University
Tehran,Tehran,Iran
2001-2003 M.Sc.

The University of Western Ontario
London, Ontario, Canada
2009-2015 Ph.D.

Honours and Awards:

Province of Ontario Graduate Scholarship
2013-2014

WGRS scholarship for 2009-2013,
University of Western Ontario

First talk reward, UWORCS2012
University of Western Ontario

Related Work Experience

Senior software engineer
SAP
2016-Present

Teaching Assistant
University of Western Ontario
2009 - 2013

Limited Duty Instructor
University of Western Ontario
2012-2013

Researcher
Iran Telecommunication Research Center
2004-2008

Publications:

1. Norouzi, Forough, and Michael Bauer. "Autonomic Management for Energy Efficient Data Centers." *CLOUD COMPUTING 2015* (2015): 153.

2. Norouzi, Forough, and Michael A. Bauer. "Toward an Autonomic Energy Efficient Data Center." *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*. IEEE :2012.

3. F.Norouzi, M. A. Bauer, "Compromise between energy consumption and QoS parameters", SoftCom2011:112.